▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 35

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 35

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas. Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Jorge Thomas, Técnico en Informática. OTROS LENGUAJES (ADA): Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación.

Ediciones Siglo Cultural, S.A.
Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 455 09 99. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-187-8
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

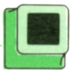
Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 259 73 31. 28020 Madrid.

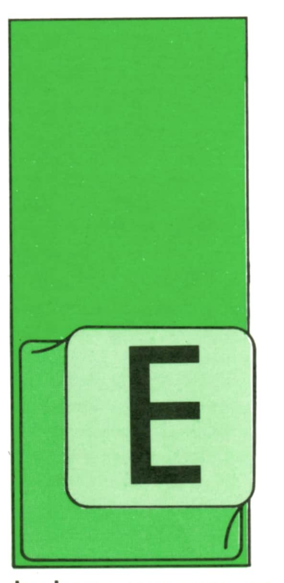
Marzo, 1988
P.V.P. Canarias: 335,–.

INDICE

BASIC



Gráficos



EL lenguaje BASIC permite la programación de gráficos de dos tipos: en baja resolución y en alta resolución.

Los gráficos en baja resolución se caracterizan porque se realizan en la pantalla de texto (pantalla de baja resolución), y en su elaboración se emplean los caracteres predefinidos en el teclado o definidos por nosotros mismos.

La pantalla de texto está dividida por una retícula que normalmente consta de 25 filas y 49 columnas, aunque puede tener otro formato, como el SPECTRUM o los MSX, cuya pantalla cuenta con 22 filas y 32 columnas.

En la figura 1 podemos ver una típica pantalla de texto.

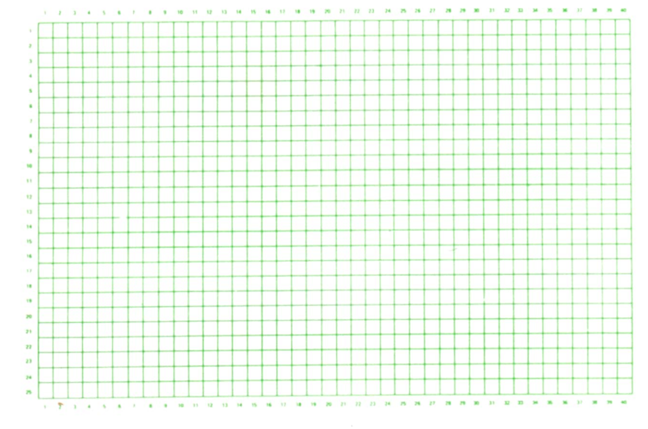




Fig. 1. Pantalla de texto de 25 filas y 40 columnas.

Realmente, ya estamos en disposición de elaborar gráficos en baja resolución, puesto que no necesitamos ninguna instrucción nueva.

De hecho, ya hicimos algunos en tomos anteriores. Sin embargo, vamos a realizar un nuevo ejemplo. El programa 1 traza en la pantalla un diagrama de barras horizontales que representa, a escala, las lluvias caídas en las distintas estaciones del año.

```
10 REM *************************
20 REM *  DIAGRAMA DE BARRAS   *
30 REM *************************
40 CLS
50 DIM L(4):DIM B(4):DIM E$(4)
60 INPUT "LLUVIA CAIDA EN PRIMAVERA";L(1)
70 INPUT "LLUVIA CAIDA EN VERANO";L(2)
80 INPUT "LLUVIA CAIDA EN OTOÑO";L(3)
90 INPUT "LLUVIA CAIDA EN INVIERNO";L(4)
100 CLS
110 LET M=L(1)
120 FOR I=2 TO 4
130 IF L(I)>M THEN LET M=L(I)
140 NEXT I
150 FOR I=1 TO 4
160 LET B(I)=L(I)*30/M
170 NEXT I
180 FOR I=1 TO 4
190 READ E$(I)
200 PRINT E$(I);TAB(11);
210 FOR J=1 TO B(I)
220 PRINT CHR$(143);
230 NEXT J
240 PRINT :PRINT :PRINT
250 NEXT I
260 DATA PRIMAVERA,VERANO,OTOÑO,INVIERNO
```

Si usamos este programa en el SPECTRUM tendremos que dimensionar la matriz alfanumérica de la línea 50 de la siguiente forma:

DIM E$ (4,9)

y sustituir la línea 160 por:

160 LET B(1) = L(1)*21/M

ya que el ancho de pantalla es más pequeño. Además, los datos de la línea DATA deben ir entre comillas.

El primer bucle FOR-NEXT del programa 1 sirve para averiguar cuál es el mayor de los cuatro datos introducidos y guardarlo en la variable M. Este dato se utiliza a continuación en el segundo bucle para hacer los cambios de escala. Por último, el tercer bucle se encarga de imprimir el diagrama en pantalla como muestra la figura 2.

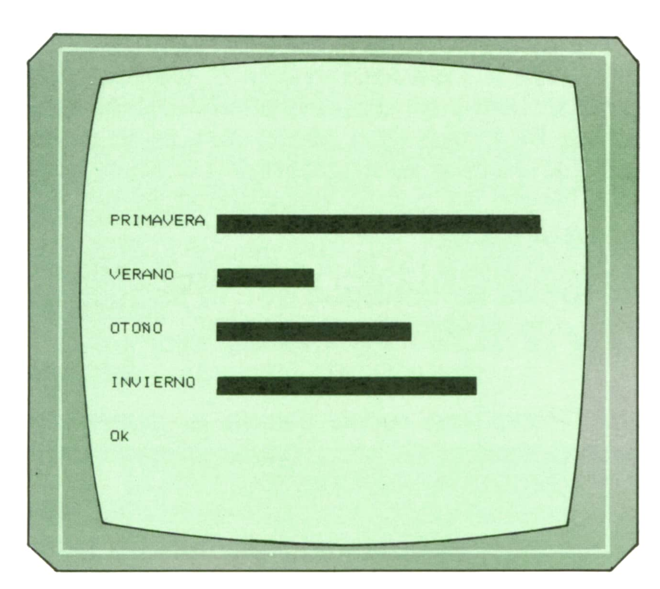
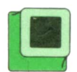



Fig. 2. Diagrama de barras obtenido con el programa 1.



Alta resolución

La pantalla de alta resolución está formada por una pequeña retícula que divide la pantalla en puntos o pixels. Normalmente una pantalla de este tipo, llamada también pantalla gráfica, tiene 200 filas y 320 columnas de pixels, aunque podemos encontrar diversos tamaños dependiendo del ordenador, tal y como podemos ver en la figura 3.

	Nº DE PUNTOS EN HORIZONTAL	Nº DE PUNTOS EN VERTICAL	OBTENCION DE LA PANTALLA
AMSTRAD	160 320 640	200	MODE Ø MODE 1 MODE 2
COMMODORE	320	200	
IBM	320 640	200	SCREEN 1 SCREEN 2
MSX	64 256	48 192	SCREEN 3 SCREEN 2
SPECTRUM	256	176	


Fig. 3. Tabla-resumen de las pantallas gráficas de los distintos ordenadores.

Un pixel se distingue de otro por sus coordenadas cartesianas. El origen de coordenadas suele estar en el ángulo inferior izquierdo, como en AMSTRAD o SPECTRUM, o en el ángulo superior izquierdo, como en IBM o MSX. Normalmente el origen es el pixel (0,0), aunque en algunos casos puede ser el (1,1). En la figura 4 podemos ver la situación de un pixel en una pantalla gráfica.

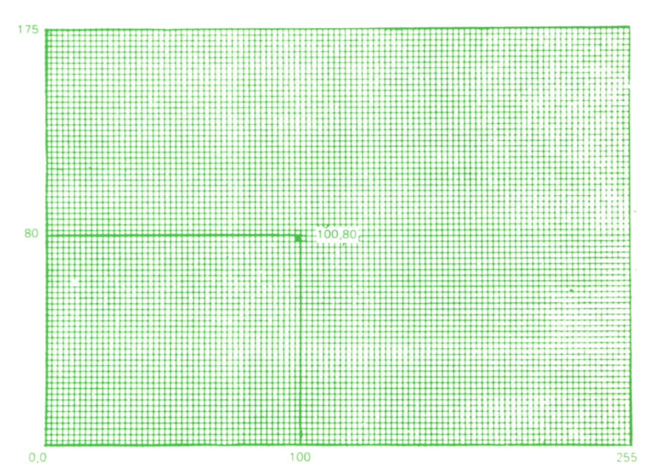



Fig. 4. Situación de un pixel en una pantalla gráfica.

Para dibujar en una pantalla gráfica, el BASIC pone a nuestra disposición una serie de instrucciones que nos permiten trazar básicamente puntos, rectas y circunferencias o arcos de circunferencia. Algunas versiones BASIC más avanzadas también permiten figuras más complejas, como elipses y cuadriláteros.

El inconveniente de los comandos gráficos es que suelen variar bastante de unas máquinas a otras, por tanto, vamos a estudiar únicamente las instrucciones fundamentales en cuatro ordenadores distintos.



Puntos

Para dibujar un punto en la pantalla gráfica el AMSTRAD y el SPECTRUM disponen de la instrucción PLOT con el siguiente formato:

PLOT x,y

siendo *x* la coordenada desde el origen hacia la derecha e *y* la coordenada desde el origen hacia arriba.

En el IBM y los MSX el trazado de un punto se consigue con la instrucción PSET, que tiene el formato siguiente:

PSET (x,y)

En este caso la coordenada *x* es la distancia desde origen a la derecha y la

coordenada *y* la distancia del origen hacia abajo.



Rectas

El trazado de rectas en el SPECTRUM se realiza con la instrucción DRAW con el siguiente formato:

DRAW x,y

donde ahora *x* representa el incremento horizontal de la recta e *y* el incremento vertical, contados ambos desde la posición en la que se encuentre el cursor gráfico. Podemos ver un ejemplo en la figura 5.

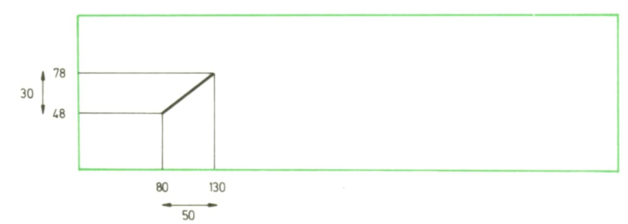




Fig. 5. Ejemplo de funcionamiento de DRAW en el SPECTRUM (DRAW 50,30).

El mismo efecto se consigue en el AMSTRAD con la instrucción DRAWR con el formato:

DRAWR x,y

En ambos ordenadores los incrementos pueden ser positivos o negativos, con los distintos efectos que muestra la figura 6.

Recta	Coordenadas (final de la recta) x	y
	+	+
	+	−
	−	−
	−	+
	Ø	+
	+	Ø
	Ø	−
	−	Ø



Fig. 6. Direcciones y sentidos de las rectas trazadas con DRAW (SPECTRUM) o DRAWR (AMSTRAD), según el signo de las coordenadas.

Por otra parte, el AMSTRAD también dispone de la instrucción DRAW, pero el funcionamiento es diferente, ya que su objetivo es trazar una recta desde la posición en la que se encuentre el cursor gráfico hasta el punto especificado en las coordenadas.

En cuanto a los MSX y el IBM, el trazado de rectas se consigue con la instrucción LINE con el formato siguiente:

LINE (X1,Y1)–(X2-Y2)

LINE traza una recta desde el punto de coordenadas (X1,Y1) hasta el punto de coordenadas (X2,Y2).

El primer punto se puede omitir, en cuyo caso el ordenador tomará la posición en la que se encuentre el cursor gráfico.

Ahora que ya sabemos dibujar puntos y rectas, vamos a ver un ejemplo. El programa 2 permite trazar todo tipo de polígonos regulares en el centro de la pantalla.

```
10 REM *************************
20 REM *  POLIGONOS REGULARES  *
30 REM *************************
40 CLS
50 INPUT "NUMERO DE LADOS";N:IF N<3 THEN GOTO 50
60 INPUT "RADIO";R:IF R<5 OR R>90 THEN GOTO 60
70 CLS:LET C1=160:LET C2=100
80 LET X1=C1+R
90 LET Y1=C2
100 LET A=2*3.1416/N
110 PSET (X1,Y1)
120 FOR I=A TO 2*3.1416+A STEP A
130 LET X=C1+R*COS(I)
140 LET Y=C2+R*SIN(I)
150 LINE -(X,Y)
160 LET X1=X:LET Y1=Y
170 NEXT I
180 PRINT "QUIERES PROBAR OTRO POLIGONO (S/N)"
190 LET R$=INKEY$:IF R$="" THEN GOTO 190
200 IF R$="S" .OR R$="s" THEN CLS:GOTO 50
```

El programa está pensado para un IBM, pero con lo explicado hasta ahora no resulta muy difícil adaptarlo a cualquier otro ordenador.

En la figura 7 podemos ver una posible ejecución de este programa.

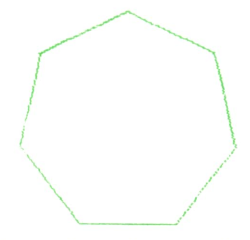
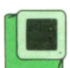




Fig. 7. Polígono trazado con el programa 2.



Circunferencias

El trazado de circunferencias se realiza en el SPECTRUM, el IBM y los MSX con la instrucción CIRCLE.

El AMSTRAD, en cambio, no dispone de ninguna instrucción para dibujar circunferencias.

El formato general de CIRCLE es el siguiente:

CIRCLE (X,Y)R

siendo *X* e *Y* las coordenadas del centro y *R* el radio. En el caso del SPECTRUM no se necesitan los paréntesis.

El programa 3 permite trazar en el centro de la pantalla todo tipo de flores de circunferencias:

```
10 REM ********************************
20 REM *  FLORES DE CIRCUNFERENCIAS  *
30 REM ********************************
40 CLS
50 INPUT "NUMERO DE PETALOS";N:IF N<3 THEN GOTO 50
60 INPUT "RADIO PRINCIPAL";R1:IF R1<5 OR R1>85 THEN GOTO 60
70 INPUT "RADIO DE LOS PETALOS";R2:IF R2<5 OR R2>90-R1 THEN GOTO 70
80 CLS:LET C1=160:LET C2=100
90 LET A=2*3.1416/N
100 AN=0
110 FOR I=1 TO N
120 X=C1+R1*COS(AN)
130 Y=C2+R1*SIN(AN)
140 CIRCLE (X,Y),R2
150 LET AN=AN+A
160 NEXT I
170 PRINT "QUIERES PROBAR OTRA FLOR"
180 LET R$=INKEY$:IF R$="" THEN GOTO 180
190 IF R$="S" OR R$="s" THEN CLS:GOTO 50
```

En las figuras 8, 9 y 10 podemos ver distintos ejemplos de la ejecución de este programa.

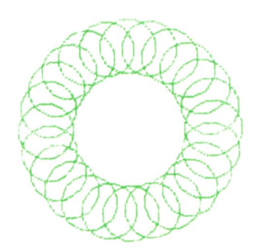




Fig. 8. Flor de circunferencias trazada con el programa 3.

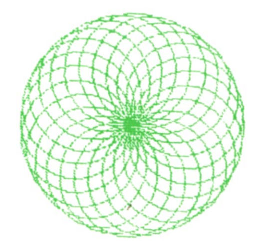




Fig. 9. Flor de circunferencias trazada con el programa 3.

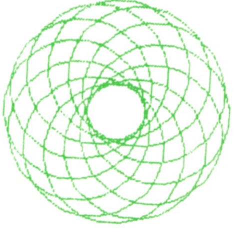
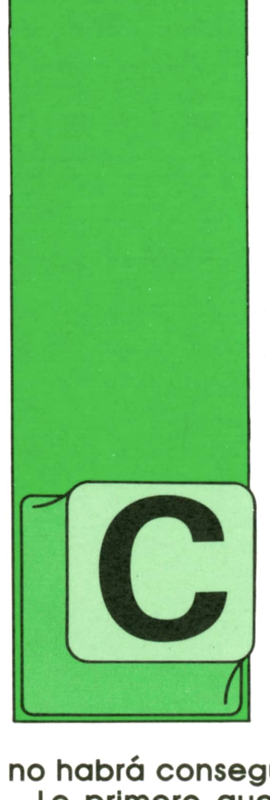




Fig. 10. Flor de circunferencias trazada con el programa 3.

Para finalizar, únicamente añadir que la mayoría de los ordenadores tratados tienen muchas más posibilidades gráficas, imposibles de tratar aquí. Nosotros sólo hemos visto lo fundamental, pero los aficionados que quieran sacar el máximo partido a los gráficos de su ordenador pueden consultar en sus manuales para programar nuevos gráficos.





Programación de las teclas de función

COMO habrá notado, su C-64 posee ocho teclas de función situadas al lado derecho y puestas una debajo de otra. Seguramente las habrá pulsado una y otra vez y no habrá conseguido nada.

Lo primero que uno puede pensar es que no funcionan, o bien, que no se está haciendo lo correcto al utilizarlas.

Bien, pues ninguna de las dos conclusiones es correcta. Lo que hay que hacer es programarlas para que ejecuten una rutina previamente introducida. Esta rutina puede ser simplemente la visualización en pantalla de una palabra, o puede hacer que el ordenador ejecute algo más complejo mediante un comando JSR al ser pulsada la tecla correspondiente.

¿Cómo hacer esto? Utilizando la técnica de interrupciones que hemos aprendido. Se intercepta la rutina, se añade una comprobación del teclado (teclas de función) y se imprime o ejecuta «algo» según la tecla pulsada.

Esto es lo que hace el programa que vamos a crear en este capítulo, que será comentado en todos los puntos en los que sea necesario.

Lo primero que hacemos es desviar las interrupciones a la zona de memoria que nos interesa. Si empezamos en la posición $C000, podemos desviar a $C010. A continuación se ofrece el listado completo tal y como aparecería en el desensamblador, pero al teclearlo basta con introducir las dos columnas de la derecha, es decir, los comandos y sus operandos cuando los haya.

```
C000 78         SEI
C001 A9 10      LDA #$10
C003 8D 14 03   STA $0314
C006 A9 C0      LDA #$C0
C008 8D 15 03   STA $0315
C00B 58         CLI
C00C 60         RTS
C00D EA         NOP
C00E EA         NOP
C00F EA         NOP
C010 48         PHA
C011 8A         TXA
C012 48         PHA
C013 98         TYA
C014 48         PHA
C015 A5 C5      LDA $C5
C017 C5 FB      CMP $FB
C019 F0 51      BEQ $C06C
C01B 85 FB      STA $FB
C01D C9 03      CMP #$03
C01F D0 08      BNE $C029
C021 A9 30      LDA #$30
C023 8D 00 C1   STA $C100
C026 4C 4A C0   JMP $C04A
C029 C9 04      CMP #$04
C02B D0 08      BNE $C035
C02D A9 00      LDA #$00
C02F 8D 00 C1   STA $C100
C032 4C 4A C0   JMP $C04A
C035 C9 05      CMP #$05
```

```
C037 D0 08      BNE $C041
C039 A9 10      LDA #$10
C03B 8D 00 C1   STA $C100
C03E 4C 4A C0   JMP $C04A
C041 C9 06      CMP #$06
C043 D0 27      BNE $C06C
C045 A9 20      LDA #$20
C047 8D 00 C1   STA $C100
C04A AD 8D 02   LDA $028D
C04D C9 01      CMP #$01
C04F D0 08      BNE $C059
C051 AD 00 C1   LDA $C100
C054 69 08      ADC #$08
C056 8D 00 C1   STA $C100
C059 A2 00      LDX #$00
C05B AC 00 C1   LDY $C100
C05E B9 01 C1   LDA $C101,Y
C061 9D 77 02   STA $0277,X
C064 E8         INX
C065 C8         INY
C066 E0 08      CPX #$08
C068 D0 F4      BNE $C05E
C06A 86 C6      STX $C6
C06C 68         PLA
C06D A8         TAY
C06E 68         PLA
C06F AA         TAX
C070 68         PLA
C071 4C 31 EA   JMP $EA31
```

— Las líneas CO10-CO14 lo único que hacen es guardar los contenidos actuales de los registros ACU, X e Y en la pila para que luego puedan ser restaurados.

— La línea CO15 carga al ACU el valor de la última tecla pulsada que se almacena siempre en la posición de memoria $CE=197.

— Las líneas CO17-CO19 efectúan una comparación entre las posiciones $CE (197) y $FB (251). Si no se ha pulsado ninguna tecla tendrán el mismo número ($40=64) y la rutina bifurca a la posición CO6C, restaurándose los contenidos de los registros que se encuentran almacenados en la pila.

Si se ha pulsado una tecla entonces los contenidos de las posiciones $C5 y $FB serán diferentes y la rutina de decodificación continúa.

— La línea CO15 carga al ACU el valor de la última tecla pulsada que se almacena siempre en la posición de memoria $C5=197.

— Las líneas CO17-CO19 efectúan una comparación entre las posiciones $CE (197) y $FB (251). Si no se ha pulsado ninguna tecla tendrán el mismo número ($40=64) y la rutina bifurca a la posición CO6C, restaurándose los contenidos de los registros que se encuentran almacenados en la pila.

Si se ha pulsado una tecla entonces los contenidos de las posiciones $C5 y $FB serán diferentes y la rutina de decodificación continúa.

— La línea CO1B coloca el valor de la tecla pulsada en $FB (251).

— La línea CO1D compara este número con el número 3 correspondiente a la tecla F7/F8.

— La línea CO1F bifurca si no se ha pulsado esta tecla, y continúa si se pulsó.

— La línea CO21 coloca en el ACU el número $30=48, que corresponde al número «0» según el código de pantalla. Este número se almacena en la posición C100 (49408), que es la posición a partir de la cual se van a almacenar las palabras que se imprimirán al pulsar las diferentes teclas de función.

— A continuación la línea CO32 salta a la posición CO4A, donde está en subrutina capaz de distinguir si la tecla pulsada fue F7 o F8. Para ello consulta la posición de memoria $028D (653) y si su contenido es 1, es que se pulsó *SHIFT.Z*Si es diferente de 1, la tecla *SHIFT* no ha sido pulsada.

Supongamos que no se pulsó SHIFT. Entonces se salta a la posición CO59, se carga el registro X con cero, pues va a servir de contador, y el registro Y con el número $30=48. La explicación de esto es muy sencilla. Si suponemos ocho letras por palabra para cada tecla, como se pulsó F7, se debe comenzar a leer en la posición 8 × 6 = 48 después de la primera que está en C100=4909.

Ahora las líneas CO5E-CO65 leen letra a letra la palabra correspondiente (LDA $C191,Y) y la colocan en el buffer del teclado (STA $0277,X). Si se pulsó SHIFT, entonces se trata de la tecla F8, con lo cual en vez de empezar a leer en la posición

48, se empezará a leer en la 56. De ello se encarga la línea CO54 (ADC=/=$08), y la CO56, que coloca en la posición C100 el número 56.

Una vez que la palabra está en el buffer del teclado y se ha comprobado que tiene 8 caracteres, se imprime en pantalla, guardándose en el registro X el número de caracteres en el buffer.

— Las líneas CO2B-C035, CO37-C041 y CO43-CO45 comprueban igualmente si la tecla pulsada fue F1/F2, F3/F4 o F5/F6, respectivamente.

— La última línea es JMP $EA31, como siempre se acaban las rutinas basadas en las interrupciones.

Por último, necesitamos los datos de las letras que se van a poner en cada tecla. Estos pueden introducirse mediante un sencillo programa en BASIC.

```
100  FOR A=0 TO 7:READ K$
110  FOR B=1 TO 8:L=ASC(MID$(K$,B,1))
120  IF L=95 THEN L=13
130  IF L=42 THEN L=4
140  POKE 49409+(A*8)+B,L:NEXT:NEXT:POKE 49409,4
150  SYS 49152
160  DATA LIST← ***
170  DATA PALABRA 2
180  DATA PALABRA 3
190  DATA PALABRA 4
200  DATA PALABRA 5
210  DATA PALABRA 6
220  DATA PALABRA 7
230  DATA PALABRA 8
```

Como cada palabra debe contener ocho caracteres, si se usan menos se debe rellenar el resto con * o el símbolo que desee sin más que cambiar su código en la línea 130.

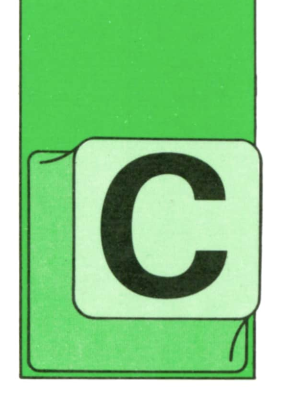
Para que la palabra lleve un retorno de carro se la hace terminar en «flecha izquierda».

Pruebe todas las palabras que se le ocurran, y si se atreve, puede ampliar la rutina para que imprima palabras clave con el resto de las teclas de su ordenador. El procedimiento es el mismo que el del ejemplo.

PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS

Movimiento rectilíneo y circular en el SPECTRUM



CON este nuevo programa vamos a poder estudiar y repasar todos nuestros conocimientos sobre el movimiento rectilíneo y circular, ya sean uniformes o acelerados.

El programa ha sido realizado de manera que todas las elecciones sobre el tipo de información que conocemos y sobre la información que deseamos obtener aparezca mediante menús de fácil uso.

Cuando nos encontremos en un menú y queramos elegir una de las opciones que se nos ofrecen, sólo tenemos que pulsar el número que se encuentra junto a la misma. Al hacerlo, dicho número empezará a parpadear. Si nos hemos equivocado de opción podemos hacerlo en ese momento pulsando, sin más, la que realmente queremos. Una vez que estemos seguros de que la opción que parpadea es la que deseamos, pulsando el número que se encuentra al lado de la frase: GOTO OPCION, ejecutamos dicha opción.

```
  1 REM ************************************
  2 REM * MOVIMIENTO RECTILINIO Y CIRCULAR *
  3 REM ************************************
  4 REM
  5 REM ********************************
  6 REM * (c) Ediciones Siglo Cultural *
  7 REM * (c) 1987                     *
  8 REM ********************************
  9 REM
 10 LET A=0
 20 LET V=0
 30 LET T=0
 40 LET L$=""
 50 LET M$="5"
 60 LET OP=1
 61 CLS
 62 PRINT  INVERSE 1;"                                "
 63 PRINT  INVERSE 1;"MOVIMIENTO RECTILINIO Y CIRCULAR"
 64 PRINT  INVERSE 1;"                                "
 65 PRINT  INVERSE 1;"         MENU PRINCIPAL         "
 66 PRINT
 70 FLASH 1
 80 PRINT "1";
 90 FLASH 0
100 PRINT ".MOVIMIENTO RECTILINEO UNIFORME"
110 PRINT "2.MOVIMIENTO RECTILINEO VARIADO"
```

```
120 PRINT "3.MOVIMIENTO CIRCULAR UNIFORME"
130 PRINT "4.MOVIMIENTO CIRCULAR VARIADO"
140 PRINT "5.GOTO OPCION"
141 PRINT : PRINT : PRINT
142 PRINT "PULSA LOS NUMEROS DEL 1 AL 5"
143 PRINT "----------------------------"
150 GO SUB 170
160 GO TO OP*1000
170 LET K$=INKEY$
180 IF L$=K$ THEN  GO TO 170
190 LET L$=K$
210 IF K$>"0" AND K$<M$ THEN  GO TO 230
220 GO TO 290
230 GO SUB 320
240 BEEP .01,30
250 LET OP=(VAL K$ AND K$<>M$)
260 FLASH 1
270 PRINT AT OP+4,0;K$
280 FLASH 0
290 IF K$<>M$ THEN  GO TO 170
300 LET O$=M$
310 RETURN
320 FOR B=0 TO (VAL M$)-1
330 PRINT AT B+5,0;B+1
340 NEXT B
350 RETURN
360 LET NV=Q/(2*PI)
370 LET AN=Q-(NV*2*PI)
380 RETURN
400 RESTORE 8780
405 READ A$
410 READ B$
415 READ C$
420 PRINT A$
425 RESTORE 8750
430 READ A$
435 PRINT A$
440 INPUT W
445 CLS
450 PRINT B$
455 INPUT AA
460 CLS
465 PRINT C$
470 INPUT Q
475 LET WO=SQR (W*W-2*AA*Q)
480 LET T=(W-SQR (W*W-2*AA*Q))/AA
485 LET F$="Wo=SQR (W^2-2*@*Q)"
490 LET G$="T=(W-SQR (W^2-2*@*Q))/@"
500 CLS
505 LET ANG=AA
510 GO SUB 3150
515 PRINT "DATOS:"
520 PRINT " W=";W;" RAD/S"
525 PRINT " @=";AA;" RAD/S^2"
530 PRINT " Q=";Q;" RAD."
535 PRINT " R=";R;" M."
540 PRINT "ECUACIONES EMPLEADAS:"
545 PRINT " ";F$'" ";G$
550 PRINT "RESULTADOS:"
555 PRINT " Wo=";WO;" RAD/S"
560 PRINT " T=";T;" SEG."
565 GO SUB 9065
570 GO TO 9095
600 RESTORE 8720
605 READ A$
610 PRINT A$
615 INPUT T
```

```
620 RESTORE 8780
625 READ A$
630 PRINT A$
635 RESTORE 8750
640 READ A$
645 PRINT A$
650 INPUT W
655 RESTORE 8800
660 READ A$
665 PRINT A$
670 INPUT Q
675 LET WO=(2*Q-T*W)/T
680 LET AA=(2*(W*T-Q))/(T*T)
685 LET F$="Wo=(2*Q-T*W)/T"
690 LET G$="@=(2*(W*T-Q))/(T^2)"
695 CLS
700 LET ANG=AA
705 GO SUB 3150
710 PRINT "DATOS:"
715 PRINT " W=";W;" RAD/S"
720 PRINT " T=";T;" SEG."
725 PRINT " Q=";Q;" RAD."
730 PRINT " R=";R;" M."
735 PRINT "ECUACIONES EMPLEADAS:"
740 PRINT " ";F$'" ";G$
745 PRINT "RESULTADOS:"
750 PRINT " Wo=";WO;" RAD/S"
755 PRINT " @=";AA;" RAD/S^2"
760 GO SUB 9065
765 GO TO 9095
800 RESTORE 8720
805 READ A$
810 PRINT A$
815 INPUT T
820 RESTORE 8780
825 READ A$
827 CLS
830 PRINT A$
835 RESTORE 8740
840 READ B$
845 PRINT B$
850 INPUT WO
860 READ B$
865 CLS
870 PRINT A$
875 PRINT B$
880 INPUT W
885 LET AA=(W-WO)/T
890 LET Q=(T/2)*(W*W-WO*WO)/(W-WO)
895 LET F$="@=(W-Wo)/T"
900 LET G$="Q=(T/2)*(W^2-Wo^2)/(W-Wo)"
905 LET ANG=AA
910 CLS
915 GO SUB 3150
920 PRINT "DATOS:"
925 PRINT " T=";T;" SEG."
930 PRINT " W=";W;" RAD/S"
935 PRINT " Wo=";WO;" RAD/S"
940 PRINT " R=";R;" M."
945 PRINT "ECUACIONES EMPLEADAS:"
950 PRINT " ";F$'" ";G$
955 PRINT "RESULTADOS:"
960 PRINT " @=";AA;" RAD/S^2"
965 PRINT " Q=";Q;" RAD."
970 GO SUB 9065
975 GO TO 9065
```

```
1000 LET M$="5"
1010 CLS
1011 PRINT  INVERSE 1;"          MENU DEL ...           "
1012 PRINT  INVERSE 1;"                                 "
1013 PRINT  INVERSE 1;" MOVIMIENTO RECTILINIO UNIFORME "
1014 PRINT  INVERSE 1;"                                 "
1015 PRINT
1020 LET OP=1
1030 FLASH 1
1040 PRINT "1";
1050 FLASH 0
1060 PRINT ".INCOGNITA 'V=VELOCIDAD'"
1070 PRINT "2.INCOGNITA 'S=ESPACIO'"
1080 PRINT "3.INCOGNITA 'T=TIEMPO'"
1090 PRINT "4.GOTO MENU PRINCIPAL"
1100 PRINT "5.GOTO OPCION"
1110 GO SUB 170
1120 CLS
1130 GO TO 1000+200*OP
1200 RESTORE 8770
1210 READ A$
1220 PRINT A$
1230 INPUT S
1240 RESTORE 8720
1250 READ A$
1260 CLS
1270 PRINT A$
1280 INPUT T
1290 CLS
1300 LET V=S/T
1310 PRINT "DATOS:"
1320 PRINT " S=";S;" M."
1330 PRINT " T=";T;" SEG."
1340 PRINT "ECUACION EMPLEADA:"
1350 PRINT " V=S/T"
1360 PRINT "RESULTADO:"
1365 PRINT " V=";V;" M."
1370 FLASH 1
1375 PRINT "PULSA UNA TECLA."
1380 FLASH 0
1385 GO SUB 1810
1390 PAUSE 0
1395 GO TO 1010
1400 RESTORE 8720
1410 READ A$
1420 PRINT A$
1430 INPUT T
1440 READ A$
1450 CLS
1460 PRINT A$
1470 INPUT V
1480 CLS
1490 LET S=V*T
1500 PRINT "DATOS:"
1510 PRINT " V=";V;" M/S"
1520 PRINT " T=";T;" S"
1530 PRINT "ECUACION EMPLEADA:"
1540 PRINT " S=V*T"
1550 PRINT "RESULTADO:"
1560 PRINT " S=";S;" M"
1570 GO TO 1370
1600 RESTORE 8770
1610 READ A$
1620 PRINT A$
1630 INPUT S
1640 RESTORE 8730
1650 READ A$
```

```
1660 CLS
1670 PRINT A$
1680 INPUT V
1690 CLS
1700 LET T=S/V
1710 PRINT "DATOS:"
1720 PRINT " S=";S;" M"
1730 PRINT " V=";V;" M/S"
1740 PRINT "ECUACION EMPLEADA:"
1750 PRINT " T=S/V"
1760 PRINT "RESULTADO:"
1770 PRINT " T=";T;" S"
1780 GO TO 1370
1800 GO TO 60
1810 PLOT 0,27
1820 DRAW 255,0
1830 OVER 1
1840 PRINT AT 18,1;"¦"
1850 PRINT AT 19,0;"T=0"
1860 PRINT AT 18,21;"¦"
1870 PRINT AT 19,17;"T=";T
1880 PRINT AT 20,10;"S=";S
1890 OVER 0
1900 PLOT 64,27
1910 DRAW 0,24
1920 DRAW 36,0
1930 DRAW 0,-24
1940 PLOT 100,40
1950 DRAW 36,0
1960 DRAW -4,2
1970 DRAW 0,-4
1980 DRAW 4,2
1990 PRINT AT 15,15;"V=";V
1995 RETURN
2000 LET M$="4"
2010 CLS
2011 PRINT  INVERSE 1;"          MENU DEL ...
2012 PRINT  INVERSE 1;"
2013 PRINT  INVERSE 1;" MOVIMIENTO RECTILINIO VARIADO.
2014 PRINT  INVERSE 1;"
2015 PRINT
2020 LET OP=1
2030 FLASH 1
2040 PRINT "1";
2050 FLASH 0
2060 PRINT ".INPUT INCOGNITAS"
2070 PRINT "2.PRINT REPRESENTACION"
2080 PRINT "3.GOTO MENU PRINCIPAL"
2090 PRINT "4.GOTO OPCION"
2100 GO SUB 170
2110 CLS
2120 GO SUB 2000+OP*200
2130 GO TO 2000
2200 GO SUB 2265
2210 IF OP=6 THEN  RETURN
2220 LET A1=OP
2230 GO SUB 2265
2235 IF OP=6 THEN  RETURN
2240 LET A2=OP
2245 IF A1=OP THEN  GO TO 2230
2250 GO SUB 8500
2255 CLS
2260 GO TO 5000+(FO-1)*300
2265 LET OP=1
2270 CLS
2271 PRINT  INVERSE 1;"
2272 PRINT  INVERSE 1;"      INTRODUCCION DE DATOS
2273 PRINT  INVERSE 1;"
```

```
2274 PRINT : PRINT
2280 LET M$="7"
2290 FLASH 1
2300 PRINT "1";
2310 FLASH 0
2320 PRINT ".VELOCIDAD FINAL 'V'"
2330 PRINT "2.VELOCIDAD INICIAL 'Vo'"
2340 PRINT "3.ACELERACION 'A'"
2350 PRINT "4.TIEMPO 'T'"
2360 PRINT "5.ESPACIO 'S'"
2370 PRINT "6.GOTO SUBMENU"
2380 PRINT "7.ELEGIR OPCION"
2390 GO SUB 170
2395 RETURN
2400 IF A=0 AND T=0 AND V=0 THEN  RETURN
2410 GO TO 8600
2600 GO TO 50
3000 LET M$="5"
3005 LET ANG=0
3010 CLS
3011 PRINT  INVERSE 1;"           MENU DEL ...           "
3012 PRINT  INVERSE 1;"                                  "
3013 PRINT  INVERSE 1;"  MOVIMIENTO CIRCULAR UNIFORME    "
3014 PRINT  INVERSE 1;"                                  "
3015 PRINT
3020 LET OP=1
3030 FLASH 1
3040 PRINT "1";
3050 FLASH 0
3060 PRINT ".INCOGNITA'W=VELOCIDAD ANGULAR'"
3070 PRINT "2.INCOGNITA'Q=ANGULO'"
3080 PRINT "3.INCOGNITA'T=TIEMPO'"
3090 PRINT "4.GOTO MENU PRINCIPAL"
3100 PRINT "5.GOTO OPCION"
3110 GO SUB 170
3120 CLS
3130 GO SUB 2999+250*OP
3140 GO TO 3000
3150 RESTORE 8810
3155 READ A$
3160 CLS
3165 PRINT A$
3170 INPUT R
3171 IF ANG<>0 THEN  GO TO 3200
3175 RESTORE 8790
3180 READ A$
3185 CLS
3190 PRINT A$
3195 INPUT AA
3200 LET S=R*Q
3205 LET V=R*W
3210 LET TA=R*AA
3215 LET AN=V*V/R
3220 CLS
3225 RETURN
3250 RESTORE 8800
3255 READ A$
3260 PRINT A$
3265 INPUT Q
3270 RESTORE 8720
3275 READ A$
3280 CLS
3285 PRINT A$
3290 INPUT T
3295 LET W=Q/T
3300 GO SUB 3150
3305 PRINT "DATOS:"
3310 PRINT " Q=";Q;" RAD."
```

```
3315 PRINT " T=";T;" SEG."
3320 PRINT " R=";R;" M."
3325 PRINT " @=";AA;" RAD/S^2"
3330 PRINT "ECUACION EMPLEADA:"
3335 PRINT " W=Q/T"
3340 PRINT "RESULTADO:"
3345 PRINT " W=";W;" RAD/S"
3400 PRINT " S=";S;" M."
3410 PRINT " V=";V;" M/S"
3420 PRINT " At=";TA;" M/S^2"
3430 PRINT " An=";AN;" M/S^2"
3440 FLASH 1
3450 PRINT "PULSA UNA TECLA."
3460 FLASH 0
3470 GO SUB 8000
3480 PAUSE 0
3490 RETURN
3500 RESTORE 8720
3510 READ A$
3520 PRINT A$
3530 INPUT T
3540 RESTORE 8780
3550 READ A$
3560 CLS
3570 PRINT A$
3580 INPUT W
3590 LET Q=W*T
3600 GO SUB 3150
3610 PRINT "DATOS:"
3620 PRINT " T=";T;" SEG."
3630 PRINT " W=";W;" RAD/S"
3640 PRINT " R=";R;" M."
3650 PRINT " @=";AA;" RAD/S^2"
3660 PRINT "ECUACION EMPLEADA:"
3670 PRINT " Q=W*T"
3680 PRINT "RESULTADO:"
3690 PRINT " Q=";Q;" RAD."
3700 GO TO 3400
3750 RESTORE 8780
3760 READ A$
3770 PRINT A$
3780 INPUT W
3790 RESTORE 8800
3800 READ A$
3810 CLS
3820 PRINT A$
3830 INPUT Q
3840 LET T=Q/W
3850 GO SUB 3150
3860 PRINT "DATOS:"
3870 PRINT " Q=";Q;" RAD"
3880 PRINT " W=";W;" RAD/SEG"
3890 PRINT " R=";R;" M."
3900 PRINT " @=";AA;" RAD/S^2"
3910 PRINT "ECUACION EMPLEADA:"
3920 PRINT " T=W/Q"
3930 PRINT "RESULTADO:"
3940 PRINT " T=";T;" SEG."
3950 GO TO 3400
3999 GO TO 50
4000 LET M$="4"
4005 LET ANG=0
4010 CLS
4011 PRINT  INVERSE 1;"           MENU DEL ...           "
4012 PRINT  INVERSE 1;"                                  "
4013 PRINT  INVERSE 1;"  MOVIMIENTO CIRCULAR VARIADO.    "
4014 PRINT  INVERSE 1;"                                  "
4015 PRINT
```

```
4020 LET OP=1
4030 FLASH 1
4040 PRINT "1";
4050 FLASH 0
4060 PRINT ".INPUT INCOGNITAS"
4070 PRINT "2.PRINT GRAFICA"
4080 PRINT "3.GOTO MENU PRINCIPAL"
4090 PRINT "4.ELEGIR OPCION"
4100 GO SUB 170
4110 CLS
4120 GO SUB 4000+OP*200
4130 GO TO 4000
4200 GO SUB 4265
4205 IF OP=6 THEN  RETURN
4210 LET A1=OP
4215 GO SUB 4265
4220 IF OP=6 THEN  RETURN
4225 LET A2=OP
4230 IF OP=A1 THEN  GO TO 4215
4235 GO SUB 8500
4240 CLS
4245 IF FO<6 THEN  GO TO 8900+(FO-1)*200
4250 IF FO<9 THEN  GO TO 400+(FO-6)*200
4255 GO TO 4610+(FO-8)*200
4265 LET OP=1
4270 CLS
4271 PRINT  INVERSE 1;"                                  "
4272 PRINT  INVERSE 1;"    INTRODUCCION DE INCOGNITAS    "
4273 PRINT  INVERSE 1;"                                  "
4274 PRINT : PRINT
4275 LET M$="7"
4280 FLASH 1
4285 PRINT "1";
4290 FLASH 0
4295 PRINT ".VELOCIDAD ANG. FINAL 'W'"
4300 PRINT "2.VELOCIDAD ANG. INICIAL 'Wo'"
4305 PRINT "3.ACELERACION ANGULAR '@'"
4310 PRINT "4.TIEMPO 'T'"
4315 PRINT "5.ANGULO 'Q'"
4320 PRINT "6.GOTO SUBMENU"
4325 PRINT "7.ELEGIR OPCION"
4330 GO SUB 170
4335 RETURN
4400 IF A=0 AND T=0 AND V=0 THEN  RETURN
4410 GO TO 3440
4600 GO TO 50
4610 RESTORE 8780
4615 READ A$
4620 PRINT A$
4625 RESTORE 8740
4630 READ B$
4635 PRINT B$
4640 INPUT WO
4645 CLS
4650 PRINT A$
4655 READ A$
4660 PRINT A$
4665 INPUT W
4670 RESTORE 8800
4675 READ A$
4680 CLS
4685 PRINT A$
4690 INPUT Q
4695 LET AA=(W*W-WO*WO)/(2*Q)
4700 LET T=(2*S*(W-WO))/(W*W-WO*WO)
4705 LET F$="@=(W^2-Wo^2)/(2*Q)"
4710 LET G$="T=(2*S*(W-WO))/(W^2-Wo^2)"
```

```
4715 LET ANG=AA
4720 CLS
4725 GO SUB 3150
4730 PRINT "DATOS:"
4735 PRINT " W=";W;" RAD/S"
4740 PRINT " Wo=";WO;" RAD/S"
4745 PRINT " Q=";Q;" RAD."
4750 PRINT " R=";R;" M."
4755 PRINT "ECUACIONES EMPLEADAS:"
4760 PRINT " ";F$'" ";G$
4765 PRINT "RESULTADOS:"
4770 PRINT " @=";AA;" RAD/S^2"
4775 PRINT " T=";T;" SEG."
4780 GO SUB 9065
4785 GO TO 9095
4810 RESTORE 8780
4815 READ A$
4820 READ B$
4825 PRINT A$
4830 RESTORE 8740
4835 READ C$
4840 PRINT C$
4845 INPUT WO
4850 READ C$
4855 CLS
4860 PRINT A$
4865 PRINT C$
4870 INPUT W
4875 CLS
4880 PRINT B$
4885 INPUT AA
4890 LET ANG=AA
4895 LET T=(W-WO)/AA
4900 LET Q=(W*W-WO*WO)/(2*AA)
4905 LET F$="T=(W-Wo)/@"
4910 LET G$="Q=(W^2-Wo^2)/(2*@)"
4915 CLS
4920 GO SUB 3150
4925 PRINT "DATOS:"
4930 PRINT " W=";W;" RAD/S"
4935 PRINT " Wo=";WO;" RAD/S"
4940 PRINT " @=";AA;" RAD/S^2"
4945 PRINT " R=";R;" M."
4950 PRINT "ECUACIONES EMPLEADAS:"
4955 PRINT " ";F$'" ";G$
4960 PRINT "RESULTADOS:"
4965 PRINT " T=";T;" SEG."
4970 PRINT " Q=";Q;" RAD."
4975 GO SUB 9065
4980 GO TO 9095
5000 RESTORE 8720
5010 READ A$
5020 PRINT A$
5030 INPUT T
5040 READ A$
5050 CLS
5060 PRINT A$
5070 READ A$
5080 PRINT A$
5090 INPUT VO
5100 RESTORE 8760
5110 READ A$
5120 CLS
5130 PRINT A$
5140 INPUT A
5150 LET V=VO+A*T
5160 LET S=VO*T+A*T*T/2
5180 LET F$="V=Vo+A*T"
```

```
5190 LET G$="S=Vo*T+A*T^2/2"
5200 CLS
5210 PRINT "DATOS:"
5220 PRINT " T=";T;" SEG."
5230 PRINT " Vo=";VO;" M/S"
5240 PRINT " A=";A;" M/S^2"
5250 PRINT "ECUACIONES EMPLEADAS:"
5260 PRINT " ";F$'" ";G$
5270 PRINT "RESULTADOS:"
5280 PRINT " V=";V;" M/S"
     PRINT " S=";S;" M."
5295 GO TO 8600
5300 RESTORE 8720
5310 READ A$
5320 PRINT A$
5330 INPUT T
5340 READ A$
5350 CLS
5360 PRINT A$
5370 READ A$
5380 PRINT A$
5390 INPUT VO
5400 RESTORE 8770
5410 READ A$
5420 CLS
5430 PRINT A$
5440 INPUT S
5450 LET V=(2*S-VO*T)/T
5460 LET A=2*(S-VO*T)/(T*T)
5470 LET F$="V=(2*S-VO*T)/T"
5480 LET G$="A=2*(S-VO*T)/(T^2)"
5490 CLS
5500 PRINT "DATOS:"
5510 PRINT " T=";T;" SEG."
5520 PRINT " Vo=";VO;" M/S"
5530 PRINT " S=";S;" M."
5540 PRINT "ECUACIONES EMPLEADAS:"
5550 PRINT " ";F$'" ";G$
5560 PRINT "RESULTADOS:"
5570 PRINT " V=";V;" M/S"
5580 PRINT " A=";A;" M/S^2"
5590 GO TO 8600
5600 RESTORE 8720
5610 READ A$
5620 PRINT A$
5630 INPUT T
5640 RESTORE 8760
5650 READ A$
5660 CLS
5670 PRINT A$
5680 INPUT A
5690 READ A$
5700 CLS
5710 PRINT A$
5720 INPUT S
5730 CLS
5740 LET V=(2*S+A*T*T)/(2*T)
5750 LET VO=(2*S+A*T*T-A*T*T*T)/(T*T)
5760 LET F$="V=(2*S+A*T^2)/(2*T)"
5770 LET G$="VO=(2*S+A*T^2-A*T^3)/(T^2)"
5780 PRINT "DATOS:"
5790 PRINT " T=";T;" SEG."
5800 PRINT " A=";A;" M/S^2"
5810 PRINT " S=";S;" M."
5820 PRINT "ECUACIONES EMPLEADAS:"
5830 PRINT " ";F$'" ";G$
5840 PRINT "RESULTADOS:"
```

```
5850 PRINT " V=";V;" M/S"
5860 PRINT " Vo=";VO;" M/S"
5870 GO TO 8600
5900 RESTORE 8730
5910 READ A$
5920 PRINT A$
5930 READ A$
5940 PRINT A$
5950 INPUT VO
5960 RESTORE 8760
5970 READ A$
5980 CLS
5990 PRINT A$
6000 INPUT A
6010 READ A$
6020 CLS
6030 PRINT A$
6040 INPUT S
6050 LET V=SQR (2*A*S+VO*VO)
6060 LET T=(SQR (ABS (2*A*S+VO*VO))-VO)/A
6070 LET F$="V=SQR (2*A*S+Vo^2)"
6080 LET G$="T=(SQR (ABS (2*A*S+Vo^2))-Vo)/A"
6090 PRINT "DATOS:"
6100 PRINT " Vo=";VO;" M/S"
6110 PRINT " A=";A;" M/S^2"
6120 PRINT " S=";S;" M."
6130 PRINT "ECUACIONES EMPLEADAS:"
6140 PRINT " ";F$'" ";G$
6150 PRINT "RESULTADOS:"
6160 PRINT " T=";T;" SEG."
6170 PRINT " V=";V;" M/S"
6180 GO TO 8600
6200 RESTORE 8720
6210 READ A$
6220 PRINT A$
6230 INPUT T
6240 READ A$
6250 CLS
6260 PRINT A$
6270 RESTORE 8750
6280 READ A$
6290 PRINT A$
6300 INPUT V
6310 READ A$
6320 CLS
6330 PRINT A$
6340 INPUT A
6350 CLS
6360 LET VO=A*T-V
6370 LET S=3*A*T*T/2-V*T
6380 LET F$="Vo=A*T-V"
6390 LET G$="S=3/2*A*T^2-V*T"
6400 PRINT "DATOS:"
6410 PRINT " T=";T;" SEG."
6420 PRINT " V=";V;" M/S"
6430 PRINT " A=";A;" M/S^2"
6440 PRINT "ECUACIONES EMPLEADAS:"
6450 PRINT " ";F$'" ";G$
6460 PRINT "RESULTADOS:"
6470 PRINT " Vo=";VO;" M/S"
6480 PRINT " S=";S;" M."
6490 GO TO 8600
6500 RESTORE 8730
6510 READ A$
6520 PRINT A$
6530 RESTORE 8750
6540 READ A$
6550 PRINT A$
```

```
6560 INPUT V
6570 READ A$
6580 CLS
6590 PRINT A$
6600 INPUT A
6610 READ A$
6620 CLS
6630 PRINT A$
6640 INPUT S
6650 LET T=(V-SQR (ABS V*V-2*A*S))/A
6660 LET VO=SQR ABS (V*V-2*A*S)
6670 LET F$="T=(V-SQR (ABS V^2-2*A*S))/A"
6680 LET G$="Vo=SQR ABS (V^2-2*A*S)"
6690 CLS
6700 PRINT "DATOS:"
6710 PRINT " V=";V;" M/S"
6720 PRINT " A=";A;" M/S^2"
6730 PRINT " S=";S;" M."
6740 PRINT "ECUACIONES EMPLEADAS:"
6750 PRINT " ";F$'" ";G$
6760 PRINT "RESULTADOS:"
6770 PRINT " T=";T;" SEG."
6780 PRINT " Vo";VO;" M/S"
6790 GO TO 8600
6800 RESTORE 8720
6810 READ A$
6820 PRINT A$
6830 INPUT T
6840 READ A$
6850 CLS
6860 PRINT A$
6870 RESTORE 8750
6880 READ A$
6890 PRINT A$
6900 INPUT V
6910 RESTORE 8770
6920 READ A$
6930 CLS
6940 PRINT A$
6950 INPUT S
6960 LET VO=(2*S-T*V)/T
6970 LET A=2*(V*T-S)/(T*T)
6980 LET F$="VO=(2*S-T*V)/T"
6990 LET G$="A=2*(V*T-S)/(T^2)"
7000 CLS
7010 PRINT "DATOS:"
7020 PRINT " T=";T;" SEG."
7030 PRINT " V=";V;" M/S"
7040 PRINT " S=";S;" M."
7050 PRINT "ECUACIONES EMPLEADAS:"
7060 PRINT " ";F$'" ";G$
7070 PRINT "RESULTADOS:"
7080 PRINT " Vo=";VO;" M/S"
7090 PRINT " A=";A;" M/S^2"
7095 GO TO 8600
7100 RESTORE 8720
7110 READ A$
7120 PRINT A$
7130 INPUT T
7140 READ A$
7150 CLS
7160 PRINT A$
7170 READ B$
7180 PRINT B$
7190 INPUT VO
7200 READ B$
7210 CLS
7220 PRINT A$
7230 PRINT B$
7240 INPUT V
7250 LET A=(V-VO)/T
7260 LET S=((V*V-VO*VO)*T)/(2*(V-VO))
7270 LET F$="A=(V-Vo)/T"
7280 LET G$="S=((V^2-Vo*^2)*T)/(2*(V-Vo))"
7290 CLS
7300 PRINT "DATOS:"
7310 PRINT " T=";T;" SEG."
7320 PRINT " V=";V;" M/S"
7330 PRINT " Vo=";VO;" M/S"
7340 PRINT "ECUACIONES EMPLEADAS:"
7350 PRINT " ";F$'" ";G$
7360 PRINT "RESULTADOS:"
7370 PRINT " A=";A;" M/S^2"
7380 PRINT " S=";S;" M."
7390 GO TO 8600
7400 RESTORE 8730
7410 READ A$
7420 PRINT A$
7430 READ B$
7440 PRINT B$
7450 INPUT VO
7460 READ B$
7470 CLS
7480 PRINT A$
7490 PRINT B$
7500 INPUT V
7510 RESTORE 8770
7520 READ A$
7530 CLS
7540 PRINT A$
7550 INPUT S
7560 LET A=(V*V-VO*VO)/(2*S)
7570 LET T=(2*S*(V-VO))/(V*V-VO*VO)
7580 LET F$="A=(V^2-Vo^2)/(2*S)"
7590 LET G$="T=(2*S*(V-Vo))/(V-2-Vo^2)"
7600 CLS
7610 PRINT "DATOS:"
7620 PRINT " V=";V;" M/S"
7630 PRINT " Vo=";VO;" M/S"
7640 PRINT " S=";S;" M."
7650 PRINT "ECUACIONES EMPLEADAS:"
7660 PRINT " ";F$'" ";G$
7670 PRINT "RESULTADOS:"
7680 PRINT " T=";T;" SEG."
7690 PRINT " A=";A;" M/S^2"
7695 GO TO 8600
7700 RESTORE 8730
7710 READ A$
7720 PRINT A$
7730 READ B$
7740 PRINT B$
7750 INPUT VO
7760 READ B$
7770 CLS
7780 PRINT A$
7790 PRINT B$
7800 INPUT V
7810 READ A$
7820 CLS
7830 PRINT A$
7840 INPUT A
7850 LET T=(V-VO)/A
7860 LET S=(V*V-VO*VO)/(2*A)
7870 LET F$="T=(V-Vo)/A"
7880 LET G$="S=(V^2-Vo^2)/(2*A)"
7890 CLS
```

```
7900 PRINT "DATOS:"
7910 PRINT " V=";V;" M/S"
7920 PRINT " Vo=";VO;" M/S"
7930 PRINT " A=";A;" M/S^2"
7940 PRINT "ECUACIONES EMPLEADAS:"
7950 PRINT " ";F$'" ";G$
7960 PRINT "RESULTADOS:"
7970 PRINT " T=";T;" SEG."
7980 PRINT " S=";S;" M."
7990 GO TO 8600
8000 CIRCLE 200,44,40
8010 PLOT 200,44
8020 DRAW 40,0
8030 PLOT 200,44
8040 DRAW (COS Q)*40,(SIN Q)*40
8050 FOR B=41 TO 40 STEP -1
8060 PLOT 200+B,44
8070 DRAW B*COS Q-B,B*SIN Q,Q
8080 NEXT B
8090 FOR B=8 TO 0 STEP -1
8100 PLOT 200+B,44
8110 DRAW B*COS Q-B,B*SIN Q,Q
8120 NEXT B
8130 PRINT AT 15,26;"Q"
8140 PRINT AT 14,31;"S"
8160 PRINT AT 16,29;"T=0"
8170 PRINT AT 17,27;"R"
8180 PLOT 200,84
8190 GO SUB 8300
8200 PRINT AT 11,29;"At"
8210 PLOT 240,44
8220 GO SUB 8300
8230 PRINT AT 7,26;"An"
8240 RETURN
8300 DRAW 0,32
8310 DRAW -2,-3
8320 DRAW 4,0
8330 DRAW -2,3
8340 RETURN
8500 IF A1=1 THEN  LET FO=(1 AND A2=5)+(2 AND A2=3)+(3 AND A2=2)+(4 AND A2=4)
8510 IF A1=2 THEN  LET FO=(3 AND A2=1)+(5 AND A2=5)+(6 AND A2=4)+(7 AND A2=3)
8520 IF A1=3 THEN  LET FO=(2 AND A2=1)+(8 AND A2=5)+(9 AND A2=4)+(7 AND A2=2)
8530 IF A1=4 THEN  LET FO=(4 AND A2=1)+(6 AND A2=2)+(9 AND A2=3)+(10 AND A2=5)
8540 IF A1=5 THEN  LET FO=(1 AND A2=1)+(5 AND A2=2)+(8 AND A2=3)+(10 AND A2=4)
8550 RETURN
8600 FLASH 1
8610 PRINT "PULSA UNA TECLA."
8620 FLASH 0
8630 GO SUB 1810
8640 PLOT 40,27
8650 DRAW -4,2
8660 DRAW 0,-4
8670 DRAW 4,2
8680 PRINT AT 17,0;"Vo=";VO
8690 PRINT AT 14,15;"A=";A
8700 PAUSE 0
8710 RETURN
8720 DATA "TIEMPO EN SEGUNDOS ?"
8730 DATA "VELOCIDAD EN M/S "
8740 DATA "(INIC.) ?"
8750 DATA "(FINAL) ?"
8760 DATA "ACELERACION EN M/S ?"
8770 DATA "ESPACIO EN METROS ?"
8780 DATA "VELOC. ANGULAR EN RAD/S "
8790 DATA "ACEL. ANGULAR EN RAD /S^2 ?"
8800 DATA "ANGULO EN RADIANES ?"
8810 DATA "RADIO EN METROS ?"
```

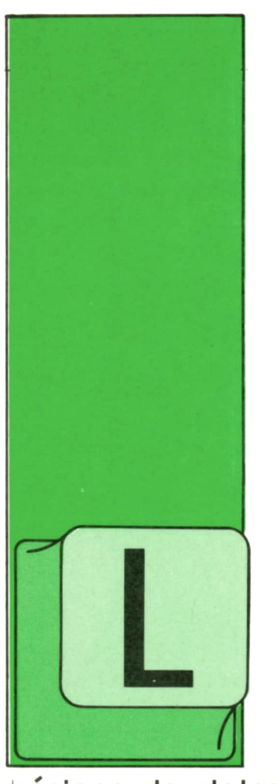
```
8900 RESTORE 8780
8905 READ A$
8910 PRINT A$
8915 RESTORE 8740
8920 READ A$
8925 PRINT A$
8930 INPUT WO
8935 RESTORE 8790
8940 READ A$
8945 CLS
8950 PRINT A$
8955 INPUT AA
8957 LET ANG=AA
8960 RESTORE 8720
8965 READ A$
8970 CLS
8975 PRINT A$
8980 INPUT T
8985 LET W=WO+AA*T
8990 LET Q=WO*T+(AA*T*T)/2
8995 LET F$="W=Wo+@*T"
9000 LET G$="Q=Wo*T+(@*T^2)/2"
9005 CLS
9010 GO SUB 3150
9015 PRINT "DATOS:"
9020 PRINT " Wo=";WO;" RAD/S"
9025 PRINT " @=";AA;" RAD/S^2"
9030 PRINT " T=";T;" SEG."
9035 PRINT " R=";R;" M."
9040 PRINT "ECUACIONES EMPLEADAS:"
9045 PRINT " ";F$'" ";G$
9050 PRINT "RESULTADOS:"
9055 PRINT " W=";W;" RAD/S"
9060 PRINT " Q=";Q;" RAD"
9061 GO SUB 9065
9062 GO TO 9095
9065 GO SUB 360
9070 PRINT "  VUELTAS=";NV;"
     + "'"  ";AN;" RAD."
9075 PRINT " V=";V;" M/S"
9080 PRINT " At=";TA;" M/S^2"
9085 PRINT " An=";AN;" M/S^2"
9090 PRINT " S=";S;" M."
9091 RETURN
9095 GO TO 3440
9100 RESTORE 8720
9105 READ A$
9110 PRINT A$
9115 INPUT T
9120 RESTORE 8780
9125 READ A$
9130 CLS
9135 PRINT A$
9140 RESTORE 8740
9145 READ A$
9150 PRINT A$
9155 INPUT WO
9160 RESTORE 8800
9165 READ A$
9170 CLS
9175 PRINT A$
9180 INPUT Q
9185 LET W=(2*Q-WO*T)/T
9190 LET AA=(2/T*T)*(Q-WO*T)
9195 LET F$="W=(2*Q-Wo*T)/T"
9200 LET G$="@=(2/T^2)*(Q-Wo*T)"
9205 CLS
9207 LET ANG=AA
9210 GO SUB 3150
9215 PRINT "DATOS:"
9220 PRINT " Wo=";WO;" RAD/S"
9225 PRINT " T=";T;" SEG."
9230 PRINT " Q=";Q;" RAD."
9235 PRINT " R=";R;" M."
9240 PRINT "ECUACIONES EMPLEADAS:"
9245 PRINT " ";F$'" ";G$
9250 PRINT "RESULTADOS:"
9255 PRINT " @=";AA;" RAD/S^2"
9260 PRINT " W=";W;" RAD/S"
9265 GO SUB 9065
9270 GO TO 3440
9300 RESTORE 8720
9305 READ A$
9310 PRINT A$
9315 INPUT T
9320 RESTORE 8790
9325 READ A$
9330 CLS
9335 PRINT A$
9340 INPUT AA
9345 READ A$
9350 CLS
9355 PRINT A$
9360 INPUT Q
9365 LET W=(2*S+AA*T*T)/(2*T)
9370 LET WO=(2*S+A*T*T-A*T*T*T)/(T*T)
9375 LET F$="W=(2*S+@*T^2)/(2*T)"
9380 LET G$="Wo=(2*S+A*T^2-A*T^3)/(T^2)"
9385 LET ANG=AA
9390 CLS
9395 GO SUB 3150
9400 PRINT "DATOS:"
9405 PRINT " T=";T;" SEG."
9410 PRINT " Q=";Q;" RAD"
9415 PRINT " @=";AA;" RAD/S^2"
9420 PRINT " R=";R;" M."
9425 PRINT "ECUACIONES EMPLEADAS:"
9430 PRINT " ";F$'" ";G$
9435 PRINT "RESULTADOS:"
9440 PRINT " W=";W;" RAD/S"
9445 PRINT " Wo=";WO;" RAD/S"
9450 GO SUB 9065
9455 GO TO 9095
9500 RESTORE 8780
9505 READ A$
9510 PRINT A$
9515 RESTORE 8740
9520 READ A$
9525 PRINT A$
9530 INPUT WO
9535 RESTORE 9070
9540 READ A$
9545 CLS
9550 PRINT A$
9555 INPUT AA
9560 READ A$
9565 CLS
9570 PRINT A$
9575 INPUT Q
9580 LET ANG=AA
9585 LET W=SQR (2*AA*Q+WO*WO)
9590 LET T=(SQR (2*AA*Q+WO*WO)-WO)/AA
9595 LET F$="W=SQR (2*@*Q+Wo^2)"
9600 LET G$="T=(SQR (2*@*Q+Wo^2)-Wo)/@"
9610 CLS
9615 GO SUB 3150
```

```
9620 PRINT "DATOS:"
9625 PRINT " Wo=";WO;" RAD/S"
9630 PRINT " @=";AA;" RAD/S^2"
9635 PRINT " Q=";Q;" RAD."
9640 PRINT " R=";R;" M."
9645 PRINT "ECUACIONES EMPLEADAS:"
9650 PRINT " ";F$'" ";G$
9655 PRINT "RESULTADOS:"
9660 PRINT " W=";W;" RAD/S"
9665 PRINT " T=";T;" SEG."
9670 GO SUB 9065
9675 GO TO 9095
9700 RESTORE 8780
9705 READ A$
9710 READ B$
9715 PRINT A$
9720 RESTORE 8750
9725 READ A$
9730 PRINT A$
9735 INPUT W
9740 CLS
9745 PRINT B$
9750 INPUT AA
9755 RESTORE 8720
9760 READ A$
9765 CLS
9770 PRINT A$
9775 INPUT T
9780 LET ANG=AA
9785 LET WO=AA*T-W
9790 LET Q=-W*T+3*AA*T*T/2
9795 LET F$="Wo=@*T-W"
9800 LET G$="Q=-W*T+(3*@*T^2)/2"
9805 CLS
9810 GO SUB 3150
9815 PRINT "DATOS:"
9820 PRINT " W=";W;" RAD/S"
9825 PRINT " @=";AA;" RAD/S^2"
9830 PRINT " T=";T;" SEG."
9835 PRINT " R=";R;" M."
9840 PRINT "ECUACIONES EMPLEADAS:"
9845 PRINT " ";F$'" ";G$
9850 PRINT "RESULTADOS:"
9855 PRINT " Wo=";WO;" RAD/S"
9860 PRINT " Q=";Q;" RAD."
9865 GO SUB 9065
9870 GO TO 9095
```

Cada vez que le digamos al ordenador que nos calcule los valores de velocidad, tiempo, espacio, aceleración, etc., para un caso particular, el ordenador también nos imprimirá la representación del movimiento.

TECNICAS DE ANALISIS

Estructura y organización de una base de datos



A estructura de la base de datos y, en consecuencia, la organización correspondiente de los datos depende del «modelo de datos» elegido. Las estructuras básicas de datos organizadas tradicionalmente de un modo jerárquico (grupos que se dividen en subgrupos), se mejoraron, en cuanto a su acceso, mediante la inclusión de tablas invertidas de índices; posteriormente se introdujeron sistemas que permitían la organización de los datos con relaciones más complejas en forma de red (sistemas que siguen básicamente las especificaciones —formuladas ya en los primeros años setenta— del Data Base Task Group encuadrado en el comité CODASYL —creador del lenguaje COBOL—) y también mediante la inclusión de la teoría de relaciones.

Aunque actualmente «están de moda» las bases de datos «relacionales», de modo que incluso algunas que no lo son gustan de definirse así, no todo son ventajas con los modelos relacionales y conviene estudiar cada caso para decidir qué tipo de estructura es la más adecuada al modelo de datos que se va a manejar.

Existen numerosos SGBD de cada uno de los tipos indicados, como se muestra en el cuadro adjunto. Veamos sucintamente cada uno de estos modelos y sus ventajas e inconvenientes:



Bases de datos jerárquicas

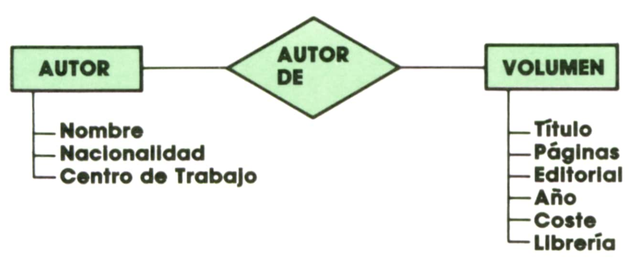
Las primeras bases de datos jerárquicas no eran más que la implementación de las estructuras de datos manejadas en COBOL, gestionadas de un modo más eficaz.

El elemento más característico de este tipo de bases de datos es la *fuerte dependencia existente en ellas entre la estructura de los datos y el modo en que físicamente están grabados* en el soporte físico.

Si queremos, por ejemplo, crear un sistema de gestión de los fondos de una biblioteca (o, sencillamente, para el manejo de nuestra biblioteca particular a nivel doméstico o profesional), deberemos establecer una estructura en que el concepto básico de organización sea el nombre del autor (para el que, también,

SGBD comerciales según la organización de datos utilizada		
ORGANIZACION	SISTEMAS COMERCIALES	
JERARQUICA	IMS	SYSTEM 2000
INDICES INVERTIDOS	ADABAS	MIISFIIT
EN RED	SOCRATES DMS IDS2	TOTAL IDMS
RELACIONALES	DATABASE 2 MRDS QBE	ORACLES INGRES

querremos tener anotados algunos otros datos complementarios: fechas de nacimiento y muerte, nacionalidad, etc.); además reseñaremos los diferentes títulos que de cada autor aparezcan en la colección, así como otros datos de cada volumen: sus características físicas, su contenido, editorial y fechas de publicación... y datos de la compra: librero a quien se adquirió el volumen, fecha, precio, etc.; por último, su situación en las estanterías. La estructura de datos resultante será como la que aparece en la figura.



ESTRUCTURA DE LA BASE DE DATOS BIBLIOGRAFICA

El contenido de la base de datos se organizará del siguiente modo:

SEGMENTO	WIRTH, Niklaus	Aleman	Inft. Fed. Tecn.			
	Alg. + Estruc. datos = Programas	382	Casti	1980	1.200	Díaz. Sant.
	The design of a Pascal Compiler	150	Amms.	1971	98FF	Lib. Univ.
SEGMENTO 2	HARTNELL, Tim	Británico				
	Int. Art.: conceptos y programas	267	Anaya	1985	1.500	Lib. Tecni.
	Libro Gig. Jueg. para ZX Spectrum	310	Anaya	1986	1.750	Edic. Perg.
	Libro Gig. Jueg. para ordenador	259	Anaya	1984	1.500	Lib. Tecni.
SEGMENTO 3	GRIES, David	Estadouni.	Cornell University			
	Compiler Cost. for Dig. Computers	452	Jhonw	1984	32$$	Thec. Lib.

Como se ve, aparece una ocurrencia por cada autor; el conjunto de elementos con él vinculados se denomina *segmento* (segmento autor); a su vez, el *segmento autor* consta de tantos *segmentos volumen* como libros de ese autor haya; en un sistema jerárquico de este tipo no se puede acceder a un segmento determinado sin acceder a su superior jerárquico, de tal modo que si queremos conocer qué libros se han comprado entre dos fechas concretas, o cuántos volúmenes hemos adquirido a un librero dado (datos ambos incluidos dentro del segmento volumen), habrá que ir accediendo sucesivamente a todos los segmentos autor para ir obteniendo de su interior los datos que deseemos.

Esta limitación es sumamente restictiva, en ocasiones, y puede producir una «ralentización» enorme de los procesos de búsqueda, si no se diseña con sumo cuidado la estructura de la base.

Además, es necesario utilizar ciertos «trucos» para resolver satisfactoriamente algunas de las tareas que, de un modo habitual, es necesario resolver en el manejo de los datos; por ejemplo:

— *Libro sin autor* (un libro que tenga diversos autores, una antología, un compendio de las comunicaciones presentadas a un congreso etc.): no se puede incluir un segmento volumen sin su correspondiente segmento autor, por lo que será necesario un autor ficticio o genérico para incluir los datos en la base de datos.

— *Datos de un librero.* Por la estructura prevista en la base, habrá que repetir los datos genéricos de un librero (domicilio, condiciones de compra, etc.) en el segmento volumen de cada libro que se haya comprado a ese librero; para evitar esta repetición inútil, se puede incluir, por ejemplo, un segmento autor para cada librero y «marcar» en algún campo separado al efecto el hecho de que los datos que siguen corresponden a una librería y no a un libro.

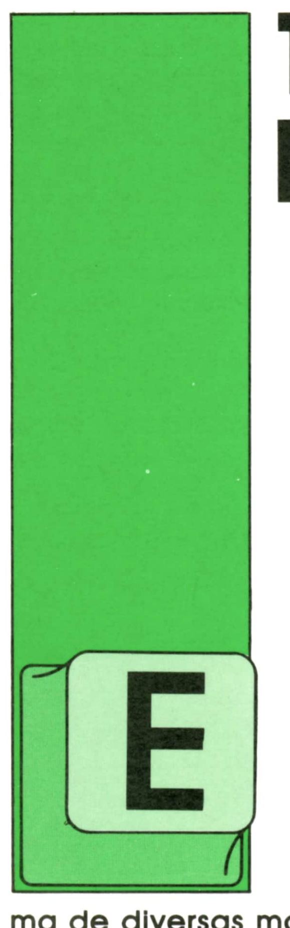
— *Actualización de datos.* Si se modifica un dato correspondiente a un librero (y el librero no está separado, como se ha indicado en el párrafo anterior), habrá que recorrer toda la base examinando dónde aparece ese librero, para ir modificando (una a una) cada aparición del librero de que se trate.

Estas situaciones descritas y otras más sutiles o sofisticadas que se presentan en la realidad, en ocasiones, muestran las dificultades y limitaciones que las bases de datos jerarquizadas pueden suponer.

TECNICAS DE PROGRAMACION



Manejo de pantalla e impresora



EN PASCAL, el manejo de la pantalla no es una característica estándar del lenguaje, por lo que los compiladores de los diversos fabricantes resuelven este problema de diversas maneras. Es preciso, por tanto, que el programador se asegure cuanto antes de cuáles son las instrucciones que puede utilizar con el compilador de que dispone. Sin embargo, las dos operaciones fundamentales, borrar la pantalla y mover el cursor a determinada posición (que corresponden a las instrucciones CLS y LOCATE de BASIC) estarán siempre disponibles de una u otra manera.

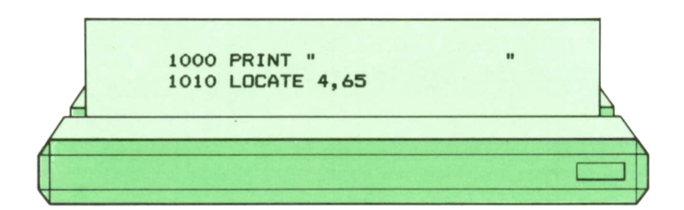
Por ejemplo, en Turbo PASCAL (uno de los compiladores de PASCAL más extendidos para ordenadores IBM PC y compatibles) estas dos instrucciones se escriben de la siguiente manera:

```
CLRSCR
GOTOXY (X, Y)
```

La primera borra la pantalla y la segunda mueve el cursor a la posición de coordenadas X (número de columna), Y (número de fila).

Turbo PASCAL tiene, además, otras funciones que actúan sobre la pantalla, como las siguientes:

1. CLREOL borra todos los caracteres desde la posición actual del cursor hasta el final de la línea, sin mover el cursor. En BASIC, esta operación podría realizarse combinando las instrucciones PRINT y LOCATE. Por ejemplo, supongamos que el cursor está, en cierto momento, en la fila 4 y la columna 65 y que queremos borrar hasta el final de la línea sin mover la posición del cursor. Podríamos conseguirlo con el programa siguiente:



En efecto, la primera instrucción (PRINT) escribe 15 espacios en blanco a partir de la posición actual del cursor (es decir, borra la línea hasta el final, pues escribiendo un espacio en blanco encima de otro carácter, éste se borra). La segunda instrucción coloca de nuevo el cursor en el lugar donde se encontraba en un principio.

2. DELLINE borra la línea donde se encuentra actualmente el cursor, moviendo el resto de la pantalla una línea hacia arriba e introduciendo una nueva línea en blanco en la parte baja de la pantalla. Esta operación no es fácil de realizar en BASIC. Para simularla, es preciso utilizar funciones avanzadas (como PEEK y POKE) y conocer con exactitud la posición de memoria donde se guarda una imagen de la pantalla.

3. INSLINE inserta una línea en blanco en la posición donde se encuentra el cursor. Las líneas que estaban situadas por debajo de éste se mueven una línea hacia abajo, y la última línea se pierde. Esta

instrucción tampoco es fácil de simular en BASIC.

Manejo de la pantalla en APL

Al igual que en PASCAL, tampoco en APL es estándar el manejo de la pantalla, por lo que suele depender del fabricante del intérprete correspondiente. En la versión de IBM, por ejemplo, existe un conjunto de funciones auxiliares que permiten realizar las operaciones siguientes:

1. Borrar la pantalla.
2. Definir «formatos» en la pantalla (es decir, dividirla en campos rectangulares como los que se explicaron en los capítulos anteriores). Pero, al revés que en BASIC y PASCAL, es posible conseguir que sea imposible escribir datos fuera de dichos campos, lo que facilita la construcción de programas que no permitan al usuario actuar fuera de los márgenes previstos por el programador. Por ejemplo, un campo puede definirse únicamente para presentación de datos o mensajes, lo que hará imposible que la persona sentada al terminal pueda cambiar la información que aparece en ese campo.
3. Hacer que el programa escriba datos en uno o más de los campos previamente definidos.
4. Colocar el cursor en cierto punto de la pantalla.
5. Interrumpir la ejecución del programa para que la persona sentada ante el terminal introduzca datos mediante el teclado en los campos en los que se le permite hacerlo. Esta operación puede funcionar como la instrucción INKEY de BASIC (devolviendo control inmediatamente y proporcionando información sobre la tecla presionada, si la hay), o bien puede incluir una espera automática hasta que se presione una tecla cualquiera, o permitir que éstas se presionen sucesivamente, cambiando la información contenida en los campos permitidos y devolviendo el control al programa sólo en el caso de que el usuario presione una de las teclas especiales (tecla ENTER, teclas de función, etc.).
6. Leer los datos contenidos en cierto momento en uno o varios campos. Esto es especialmente aplicable a los campos en los que se permite introducir datos al utilizador del programa.

Manejo de la impresora

Es muy frecuente que el ordenador personal tenga conectada una impresora, donde podemos obtener copias en papel, ya sea de la información que aparece en la pantalla, ya sea de datos cualesquiera que deseemos conservar de una forma visible y permanente. Normalmente, la impresora podrá utilizarse de tres maneras diferentes:

— Para obtener una copia del contenido de la pantalla en un momento dado. A veces, esta función se denomina «instantánea», pues equivale a la toma de una fotografía instantánea de la pantalla.

— Para obtener una copia continua, línea a línea, de todo lo que va apareciendo por la pantalla. Esto es lo que suele llamarse una «bitácora» de la sesión de trabajo, por analogía con el cuaderno de bitácora de los barcos, donde se anotan todas las incidencias que suceden durante la travesía. El término inglés correspondiente, también utilizado en informática, es «log book», que hace referencia al cuaderno utilizado en los barcos para anotar la velocidad de la nave en cada momento de la travesía.

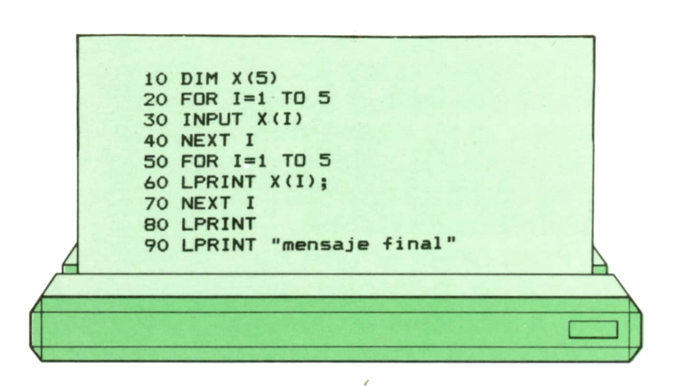
— Para enviar a la impresora, bajo el control del programa, uno o varios datos, independientemente de su aparición o no por la pantalla.

Las dos primeras funciones están, normalmente, bajo el control del sistema operativo, y se ponen en funcionamiento automáticamente al presionar ciertas teclas. Por ejemplo, en los ordenadores IBM PC y compatibles, existe una tecla (que lleva el nombre PrtSc, u otro equivalente), que al ser presionada (junto con alguna tecla especial, como la de mayúsculas o la de control) provoca automáticamente la obtención de una instantánea o la puesta en marcha de la bitácora (que se mantendrá hasta que se desabilite, presionando de nuevo la misma tecla).

La tercera función (el envío de datos a la impresora sin que aparezcan en pantalla) es propia de cada lenguaje de programación.

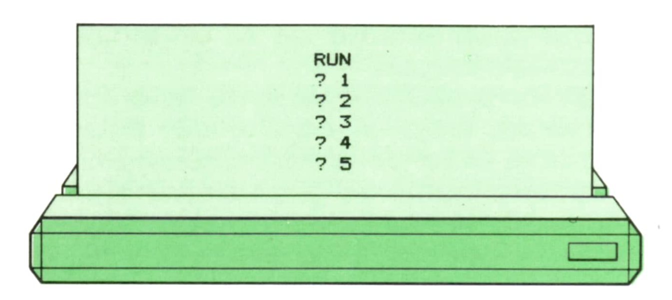
En BASIC existe la instrucción LPRINT, equivalente a PRINT, pero que se diferencia de ésta porque el texto o expresión correspondiente es enviado a la impresora en lugar de a la pantalla.

Por ejemplo, sea el programa siguiente (véase el capítulo 32, para la versión equivalente que utiliza la instrucción PRINT):

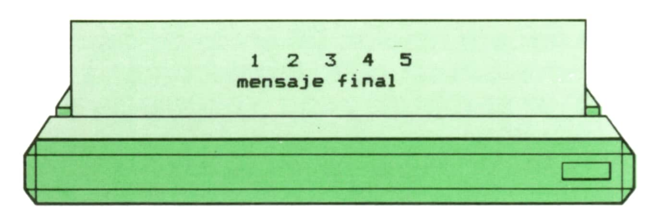


Este programa lee cinco valores del teclado, los introduce en la variable X y después envía a la impresora dichos valores (mediante un bucle), escribiéndolos en una sola línea. Finalmente, pasa a la línea siguiente y escribe un mensaje final. Obsérvese que a la instrucción LPRINT se le aplican las mismas normas que a la instrucción PRINT: la presencia o ausencia de un punto y coma al final de la expresión a imprimir decide si se añadirá o no, automáticamente, un paso al principio de la línea siguiente.

Veamos cómo aparecerá por la pantalla la ejecución del programa anterior:



Obsérvese que es idéntica a la ejecución del ejemplo del capítulo 32, excepto que los resultados de LPRINT no aparecen por la pantalla. Sí, en cambio, aparecerán por la impresora, donde se leerá lo siguiente:

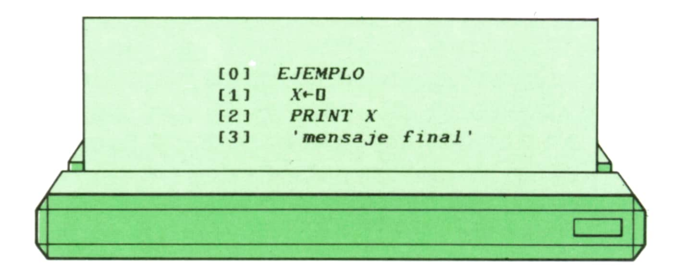


En PASCAL no existe un método estándar de manejo de la impresora, por lo que diversos compiladores pueden hacerlo de distintas formas. En Turbo PASCAL, el procedimiento consiste en utilizar las instrucciones WRITE y WRITELN, como si fuéramos a escribir en la pantalla, pero introduciendo un primer parámetro en la lista de datos a imprimir (la palabra reservada LST), que indica al compilador que se desea obtener el resultado por la impresora en lugar de por la pantalla.

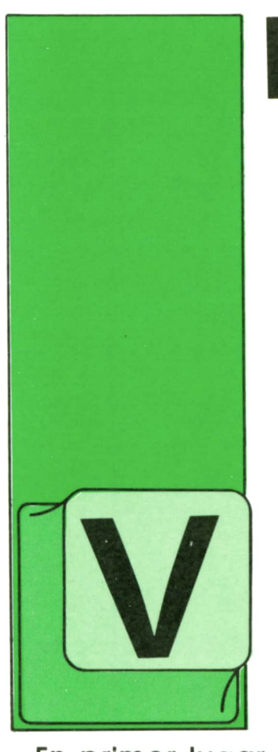
Veamos cómo se escribiría en Turbo PASCAL el programa que realiza la misma función que el que acabamos de ver en BASIC:

```
program ejemplo;
 var x:array[1..5] of integer;
     i:integer;
 begin
   for i:=1 to 5 do
       readln (x[i]);
   for i:=1 to 5 do
       write (lst,x[i],' ');
   writeln (lst,'');
   writeln (lst,'mensaje final')
 end.
```

Finalmente, tampoco en APL es estándar el manejo de la impresora. En IBM/PC APL se proporciona una función pre-programada (PRINT) que permite enviar valores a la pantalla. Veamos cómo se escribiría en APL el programa anterior:



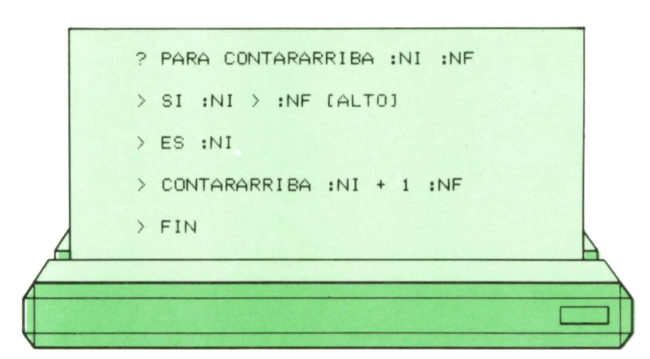
LOGO



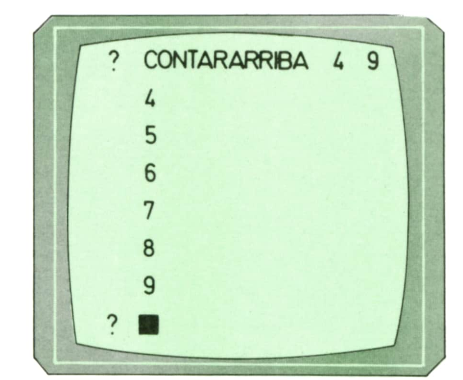
Dos procedimientos parecidos

AMOS a ver dos procedimientos recursivos cuya definición es muy parecida, pero que a la hora de que los ejecute la tortuga se obtiene un resultado chocante.

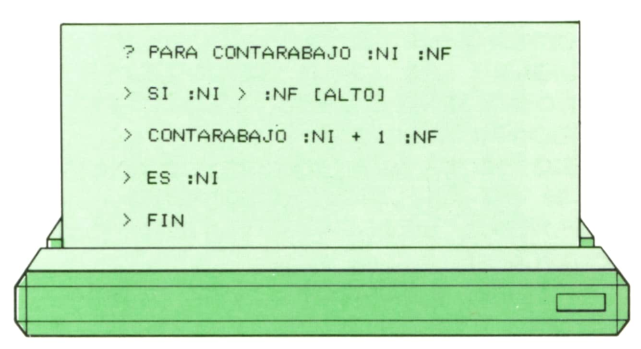
En primer lugar, definimos un procedimiento para que se escriban por pantalla un conjunto de números a partir de uno inicial (NI) hasta uno final (NF). Queda así:



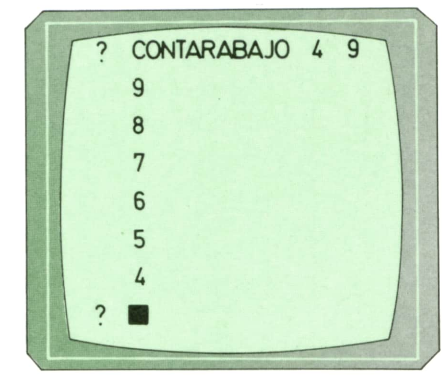
Si lo ejecutamos:



Ahora escribimos otro procedimiento cambiando el orden de dos líneas del anterior:

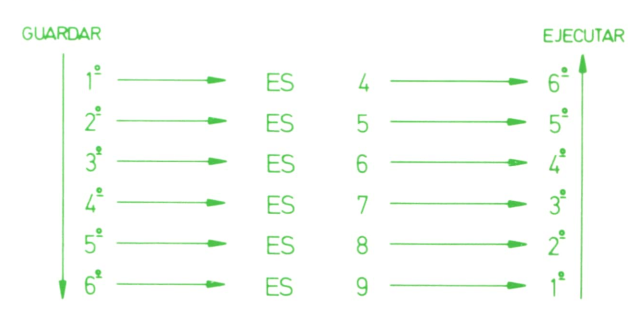


y lo ejecutamos con los mismos valores:



¡Nos queda justo lo contrario! ¿Por qué? Es muy sencillo. En este segundo caso, cada vez que vamos haciendo una llamada recursiva nos va quedando un comando sin ejecutar (el de escritura del número). Lo que hace la tortuga es ir guardando en su memoria lo que deja sin escribir para no olvidarse de hacerlo cuando termine.

Cuando llega al comando ALTO mira a ver si le queda algo pendiente y, si es así, lo ejecuta en orden inverso a como lo ha ido guardando.



Es lo mismo que si ponemos varios platos uno encima de otro en una mesa y luego los vamos quitando de uno en uno. El primero que pusimos es el último que quitamos.



Un procedimiento para hacer un reloj

Vamos a escribir procedimientos para dibujar un reloj, pero con la particularidad de que va a andar, es decir, va a ir marcando la hora.

Para hacer esto, tenemos que utilizar varias variables para lo siguiente:

— HORA, MIN y SEG, para indicar la hora inicial.

— POSIH y RUMBOH, para guardar la posición y la dirección de la aguja de la hora.

— POSIM y RUMBOM, para guardar la posición y la dirección del minutero.

— POSIS y RUMBOS, para guardar la posición y la dirección del segundero.

Además de utilizar comandos que ya hemos visto, necesitamos usar uno nuevo para que, prácticamente, el reloj funcione como uno de verdad (el segundero se mueva cada segundo, el minutero cada minuto,...). Así, el comando

ESPERA n

hace que la tortuga detenga la ejecución de un procedimiento durante n/50 segundos, siendo *n* un número. Por tanto, si queremos que espere un segundo tendremos que poner

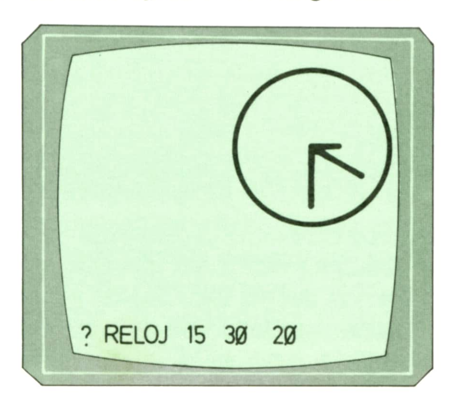
ESPERA 50

Ya podemos definir todos los procedimientos:

```
? PARA RELOJ :HORA :MIN :SEG
> BP OT
>  SI  :HORA  > 11 [HAZ "RUMBOH (:HORA - 12) * 30]  [HAZ  "RUMBOH
:HORA * 30]
> HAZ "RUMBOH :RUMBOH + :MIN * 0.5
> HAZ "RUMBOM :MIN * 6
> HAZ "RUMBOS :SEG * 6
> PONCL 10
> ESFERA
> RAYAS
> SL PONPOS [0 0] BL
> PONRUMBO :RUMBOH PONCL 8 AV 27
> HAZ "POSIH POS
> SL PONPOS [0 0] BL
> PONRUMBO :RUMBOM PONCL 8 AV 32
> HAZ "POSIM POS
> SL PONPOS [0 0] BL
> PONRUMBO :RUMBOS PONCL 3 AV 37
> HAZ "POSIS POS
> ANDA
> FIN
? PARA ESFERA
> SL AV 57 GD 90 BL
> REPITE 360 [AV 1 GD 1]
> SL PONPOS [0 0] GI 90 BL
> FIN
```

```
? PARA RAYAS
> REPITE 12 [SL AV 42 BL AV 10 SL PONPOS [0 0] GD 30 BL]
> FIN
? PARA ANDA
> REPITE 60 - :SEG [SL PONPOS [0 0] GOMA ESPERA 50 PONPOS :POSIS
SL PONPOS [0 0] BL PONCL 8 PONPOS :POSIH SL PONPOS [0 0] BL
PONPOS :POSIM SL PONPOS [0 0] BL PONRUMBO :RUMBOS GD 6 HAZ
"RUMBOS RUMBO PONCL 3 AV 37 HAZ "POSIS POS]
> SL PONPOS [0 0] GOMA
> SI :RUMBOM = :RUMBOS [PONPOS :POSIS] [PONPOS :POSIM]
> SL PONPOS [0 0] BL PONCL 3 PONPOS :POSIS
> SL PONPOS [0 0] BL PONCL 8 PONPOS :POSIH
> SL PONPOS [0 0] BL
> PONRUMBO :RUMBOM
> GD 6
> HAZ "RUMBOM RUMBO
> PONCL 8 AV 32
> HAZ "POSIM POS
> SL PONPOS [0 0] GOMA
> SI :RUMBOH = :RUMBOS [PONPOS :POSIS] [SI :RUMBOH = :RUMBOM
[PONPOS :POSIM] [PONPOS :POSIH]]
> SL PONPOS [0 0] BL PONCL 3 PONPOS :POSIS
> SL PONPOS [0 0] BL PONCL 8 PONPOS :POSIM
> SL PONPOS [0 0] BL
> PONRUMBO :RUMBOH
> GD 0.5
> HAZ "RUMBOH RUMBO
> PONCL 8 AV 27
```

Al ejecutarlo quedaría algo así:



El reloj no dejará de andar hasta que no pulsemos la tecla de parada.

Hay que tener en cuenta dos cosas. Lo primero es que para poner el reloj en hora real hay que dar un valor inicial un poco mayor, porque la tortuga tarda un tiempo en dibujar el reloj y en ponerle en hora. Lo segundo es que poco a poco el reloj se irá retrasando porque también lleva tiempo el ejecutar todos los comandos del procedimiento ANDA.



Un comando de lectura

Hasta ahora, cuando hemos querido dar valor a una variable lo hemos hecho de dos formas:

1. Poniendo la variable al lado del nombre de un procedimiento al definirlo y luego escribiendo el valor cuando lo ejecutamos.
2. Utilizando el comando de asignación HAZ.

Existe una tercera posibilidad que consiste en leer el valor para esa variable del teclado, es decir, almacenar en ella el carácter o el número correspondiente a la tecla que pulsemos.

Para eso, tenemos que usar el comando

LEECAR



o en abreviatura

LC

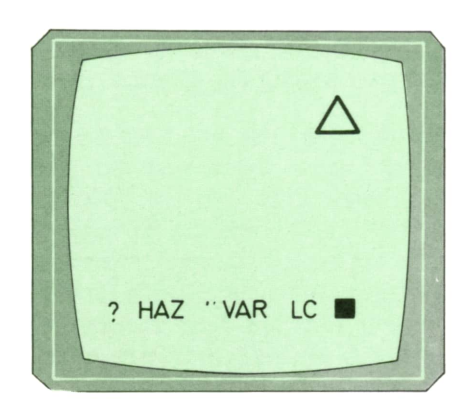
Cuando escribimos este comando y le mandamos a la tortuga que lo ejecute, ésta se queda esperando a que pulsemos una tecla.

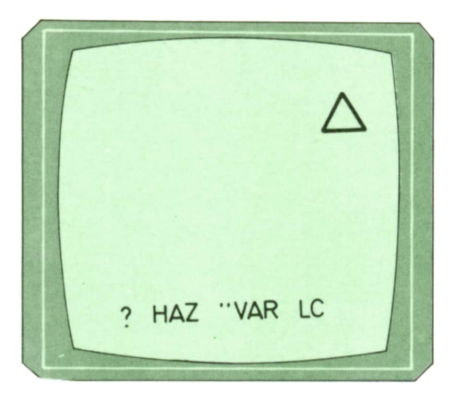
Como es lógico, el valor de la tecla lo tenemos que guardar en algún sitio. Este sitio es una variable, por lo que lo más normal es escribir

HAZ "VAR LC

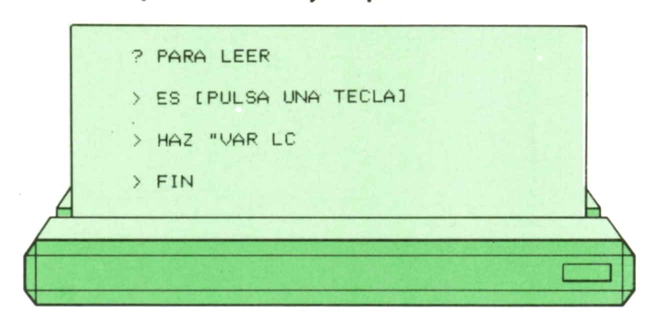
De esta forma, le decimos a la tortuga que almacene en la variable VAR el carácter (letra o número) de la tecla que pulsemos.

Podemos saber que la tortuga está esperando a que pulsemos una tecla porque en la pantalla no ocurre nada ni se ve el signo de interrogación (?) ni el cursor

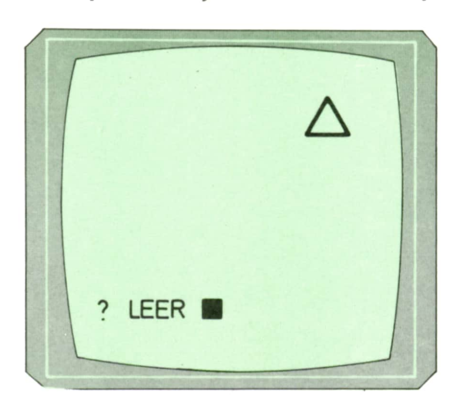




o porque nosotros mismos le digamos que escriba un mensaje para que nos sirva de ayuda. Por ejemplo así:



de forma que al ejecutarlo nos queda:

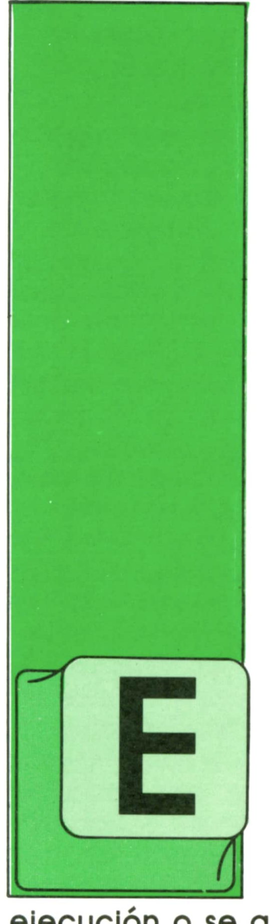




Por último, cuando pulsamos una tecla, se almacena su valor en la variable, pero éste no se ve en la pantalla. Para saber cuál es podemos escribir el contenido de dicha variable con el comando ES.

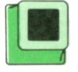
PASCAL

Almacenamiento de estructuras dinámicas en ficheros



EL programa «experto» de los animales que se ha presentado como ejemplo de aplicación de árboles tiene el inconveniente de que, cada vez que se termina su ejecución o se apaga el ordenador, todos los datos trabajosamente introducidos a golpe de tecla se pierden y hay que empezar de nuevo.

Está claro que un programa así es poco útil. Ya sabemos cómo crear ficheros de variables de cualquier tipo: sin embargo, no resulta posible definir ficheros cuyos elementos sean estructuras dinámicas, ya que éstas, como es lógico, tienen unas dimensiones y una estructura no definidas *a priori.*



Almacenamiento de listas encadenadas

El almacenamiento (y posterior recuperación) de estructuras de tipo lista encadenada en ficheros no presenta mayores problemas; basta con utilizar ficheros cuyos elementos sean del mismo tipo que los elementos de las listas e ir guardando (o recuperando) éstos de manera secuencial hasta llegar al final de la lista (o al final del fichero). Por ejemplo, para guardar en un fichero F una lista encadenada diríamos:

```
rewrite (F);
P:=Raiz;
while P < > nil do
   begin
   F^:=P^; put (F);
   p :=p^. Siguiente
   end;
```

o bien, si el compilador lo admite:

```
rewrite (F);
P:=Raiz;
while P <> nil do
   begin
   write (F, p^);
   P:= P^. Siguiente
   end;
```

Para recuperar de un fichero así creado una lista encadenada, podríamos hacer:

```
reset (F);
Raiz:=nil;
if not eof (F) then
   begin
     new (Raiz);
     P:=Raiz;
     P^:= F^; get (F);
     while not eof (F) do
      begin
        new (P^.Siguiente);
       P:=P^.Siguiente;
        P^:= F^; get (F)
      end;
     p^.Siguiente:= nil
   end;
```

Los elementos se guardan completos en el fichero, es decir, incluyendo el campo correspondiente al puntero; realmente, este campo no es necesario, ya que el valor que va a adquirir cada uno al recuperarse la lista es el asignado por el procedimiento NEW, como puede observarse en el programa. Para ahorrar ese espacio, podríamos definir los tipos involucrados de la siguiente manera:

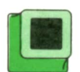
```
type
  SubFicha_t= (* véase el texto *)
  Puntero_t = ^Ficha_t;
  Ficha_t = récord
                Subficha : SubFicha_t;
                Siguiente: Puntero_t;
              end;
  Fichero_t = file of SubFicha_t
```

SubFicha_t sería un tipo que englobase a todos los campos de Ficha_t, excepto el puntero. La recuperación desde un fichero, por ejemplo, podría ser:

```
reset (F);
Raiz:=nil;
if not eof (F) then
   begin
     new (Raiz);
     P:=Raiz;
     P^.SubFicha:= F^; get (F);
     while not eof (F) do
      begin
        new (P^.Siguiente);
        P:=P^.Siguiente;
        P^.SubFicha:= f^; get (F)
      end;
     P^.Siguiente:= nil
   end;
```

y el almacenamiento:

```
rewrite (F);
P:=Raiz;
while P <> nil do
   begin
   F^:= P^.SubFicha; put (F);
   P := P^.Siguiente
   end;
```



Almacenamiento de árboles

Para guardar un árbol, hay que salvar uno por uno, secuencialmente, todos sus nodos; por lo tanto, hay que escoger alguno de los métodos de recorrido de árboles que conocemos: pre-orden, post-orden u orden central (si el árbol no es binario, puede haber más de un orden central). Con pre-orden, el procedimiento podría ser:

"Salvar el árbol encabezado por tal nodo:"

1 — Salvar el nodo en cuestión.

2 — Si tiene descendiente izquierdo, entonces "salvar el árbol encabezado por su descendiente izquierdo".

3 — Si tiene descendiente derecho, entonces "salvar el árbol encabezado por su descendiente derecho".

Para salvar el árbol había que «salvar el árbol encabezado por el nodo raíz».

Para recuperar un árbol de un fichero, se iría leyendo éste secuencialmente y reconstruyendo el árbol nodo a nodo. Los campos correspondientes a los punteros contienen las direcciones en memoria que tenían los descendientes cuando se salvó el árbol. Estas no tienen por qué ser las mismas al reconstruirlo en un momento posterior (puede ser un programa distinto, con más variables que ocupen sitio en memoria y desplacen la posición de ciertos datos...) y, como sucedía con las listas, su valor será el que se asigne con el procedimiento NEW al recuperar cada nodo.

Sin embargo, su contenido sí es útil: si el contenido de esos campos es distinto de NIL, eso quiere decir que, cuando se salvó el nodo, tras él se salvaron las ramas que de él salían, que, por tanto, se encuentran a continuación en el fichero. El procedimiento habría de reflejar, por tanto, el recorrido en pre-orden utilizado al crear el fichero:

"Recoger el árbol encabezado por tal nodo:"

1 — Reservar sitio en memoria para el nodo, dejando listo el puntero que llevará hasta él.

2 — Leer su contenido del fichero.

3 — Si existen descendientes:

- **Recoger el árbol encabezado por su descendiente izquierdo.**
- **Recoger el árbol cabezado por su descendiente derecho.**

En realidad, se puede ahorrar algo de espacio si, en lugar de guardar las fichas completas, se guardan todos los campos menos los punteros y en lugar de éstos se guardan unas variables de tipo Boolean que indiquen si existen descendientes o no.

Como aplicación de todo esto veamos cómo modificar el programa de los animales para no tener que empezar desde cero cada vez (seguiremos utilizando el Turbo-Pascal). En primer lugar, podríamos definir la constante NOMBRE, para el nombre del fichero:

```
const
  Nombre = 'ANIMALES.ARB';
  (* las demás constantes ... *)
```

y la variable FICHERO:

```
var
  Fichero: file of Ficha_t;
  (* las otras variables ... *)
```

El procedimiento de recogida de datos desde un fichero podría ser:

```
procedure Recoger (var P: Punt_t);
(* Recoge del fichero el nodo al que apuntará P y sus ramas *)
 begin
  new (P);
  read (Fichero, P^);
  with P^ do
   begin
    if QueSi <> nil then  Recoger (QueSi);
    if QueNo <> nil then  Recoger (QueNo)
   end
 end;
```

Como el dato que se pasa es el puntero que lleva al nodo, el paso de parámetros ha de ser por nombre para que, tras ejecutarse NEW, se quede realmente apuntando a la zona de memoria reservada. Tras ejecutar Reset para posicionarse al principio del fichero, se ejecutaría Recoger (Raiz).

Para salvar la información del árbol, tras ejecutar Rewrite se podría ejecutar Salvar (Raiz), siendo Salvar el siguiente procedimiento:

```
procedure Salvar (P: Punt_t);
(* Guarda en el fichero el nodo al que apunta P y sus ramas *)
 begin
  write (Fichero, P^);
  with P^ do
   begin
    if QueSi <> nil then Salvar (QueSi);
    if QueNo <> nil then Salvar (QueNo)
   end
 end;
```

Por supuesto, hay que ejecutar después el procedimiento Close. Con todo esto, sólo quedaría modificar el programa principal:

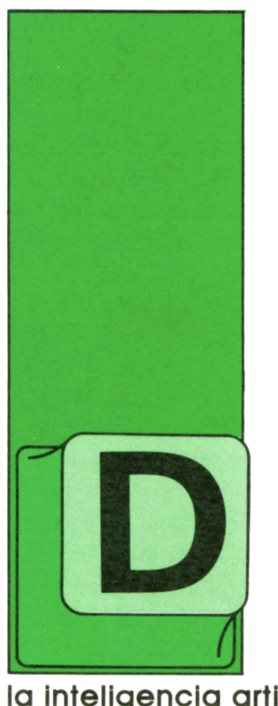
```
begin
write ('¿Leo datos de disco');
if Afirmativo then
   begin
   assign (Fichero, Nombre);
   reset (Fichero);
   Recoger (Raiz);
   close (Fichero)
   end
 else
   begin
   new (Raiz);
   with Raiz^ do
     begin
     Pregunta:='la mosca
     QueSi    := nil;
     QueNo    := nil
     end
   end;

 repeat
  writeln;
  BuscaEn (Raiz);
  writeln;
  write ('¿Desea seguir');
 until not Afirmativo;

write ('¿Guardo los datos (reemplazando al anterior árbol)');
if Afirmativo then
   begin
   assign (Fichero, Nombre);
   rewrite (Fichero);
   Salvar (Raiz);
   close (Fichero)
   end
end.
```

El programa resultante no dispone de tratamiento de errores, por lo que si, al intentar leerlo, no existiera el fichero ANIMALES.ARB, o, al intentar salvar los datos, no hubiera suficiente espacio en disco o éste estuviera protegido contra grabación, se produciría un error y se detendría el programa.

OTROS LENGUAJES





El lenguaje LISP

EL lenguaje LISP se ha hablado mucho últimamente, sobre todo debido al gran auge que ha tomado un campo, relativamente moderno de la programación, como es la inteligencia artificial.

Sin embargo, este lenguaje es realmente innovador, ya que parte de un concepto no utilizado hasta ahora, por lo que es difícil juzgar su eficacia, dentro de los cánones habituales utilizados con otros lenguajes llamados clásicos.

Se puede decir, sin embargo, que es realmente eficaz para el tratamiento de listas, de ahí su nombre: «LISt Processing». Estas listas eran originariamente frases de lenguaje natural que debían ser procesadas (interpretadas) por el ordenador. De ahí su utilidad en inteligencia artificial, que basa principalmente sus actividades en el procesado del lenguaje (al menos inicialmente).

Trataremos de dar una idea lo más aproximada posible de su funcionamiento, así como de explicar algunas ideas esenciales relacionadas con el tipo de actividades en las que se utiliza este lenguaje. De todas formas, no debemos olvidar que el LISP es un lenguaje «raro» para profanos, ya que su arquitectura no tiene mucho que ver con los lenguajes imperativos. (Como el BASIC, FORTRAN, PASCAL, COBOL, y prácticamente todos los conocidos por el gran público.)

El lenguaje LISP es definido por los teóricos muy brevemente. Ellos dicen que LISP es un lenguaje Funcional, Simbólico, Recursivo, Lógico, para el Procesado de Listas.

Cada una de estas definiciones encierra un amplio contenido, en algunos casos muy diferente del habitual, por lo que deberemos explicar cada una de ellas detalladamente.

En primer lugar, la palabra Funcional no se refiere a que sea muy eficaz, sino a algo mucho más dramático. En LISP sólo existe una sentencia (es decir, un solo tipo de sentencia), ésta es la función.

Algunos entendidos dicen que ésta es la principal causa de su belleza y consistencia.

El programador clásico se preguntará, ¿cómo es posible hacer todo un programa, con sólo llamadas a funciones; sin asignaciones, sin bucles...? ¡Horror!

Y no sólo un programa, sino los programas más monstruosamente complicados, ya que, por desgracia, la inteligencia artificial conlleva una gran complejidad de proceso con un número de datos ingente.

De todas formas, el sufrido lector no debe asustarse demasiado, ya que todas las estructuras clásicas de programación siguen conservándose en LIPS, gracias a que una función puede hacer casi cualquier cosa, como una asignación, un bucle, y todo lo demás. En realidad, podríamos casi decir que la estructura «Funcional» es casi un subterfugio teórico para dotarlo de una uniformidad que raya en lo imposible, muy útil, eso sí.

Un programa en LISP es, pues, una función, formada a su vez por otras funciones, definidas por el programador, o que provienen de la biblioteca de funciones estándar que «definen» la sintaxis del lenguaje.

Nota: Es evidente que cuando hablamos de función nos referimos al concepto clásico, muy similar al del PASCAL, por ejemplo.

La segunda palabra que define al lenguaje es Simbólico.

Esta no sólo quiere decir que se utilicen nombres de variables «abstractas» como en todos los lenguajes, sino que el tipo de los datos utilizados es, en esencia, abstracto, es decir, simbólico.

Tratemos de explicar esto más claramente.

Tan abstractos son los datos en LISP que, en realidad, sólo hay dos tipos de datos: átomos y listas. Un átomo es indivisible y podríamos explicarlo como una cadena de caracteres (string) sin espacios en blanco. El espacio en blanco en LISP, al contrario que en los lenguajes clásicos, tiene una importancia esencial, ya que es el separador por excelencia (equivale a una coma).

Además, una lista es una tira de átomos separados por espacios. Entonces, ¿cómo utilizar datos de otro tipo, como números? Pues, sencillamente, un número es un átomo al que aplicaremos funciones que consideremos adecuadas, como la suma, por ejemplo.

Debemos dejar claro otra idea fundamental del LISP. Un dato y una variable son casi lo mismo. Esto quiere decir que un átomo puede contener un valor, con lo que sería en teoría una variable, pero su contenido puede tener, a su vez, un contenido, y así hasta el infinito. Con esto, datos e identificadores son lo mismo, ya que podemos «meter» una variable con todos sus contenidos en otra, que no se diferencia en nada de un dato, ya que ambos son átomos (en todo lo anterior, donde se dice átomo, puede decirse también lista). Sencillo, ¿eh?

Con las ideas dadas hasta ahora, el lector puede darse una leve idea de la potencia «posible» del lenguaje. Sin límites entre datos y programa, no hay meta a la que un programador «original» no pueda llegar. Sin embargo, esta falta aparente de reglas provoca en el no iniciado una especie de «vértigo» debida a lo que podríamos denominar «mal de altura» debido al excesivo número de posibilidades nuevas que proporciona el lenguaje.

La tercera palabra que define al lenguaje es Recursivo.

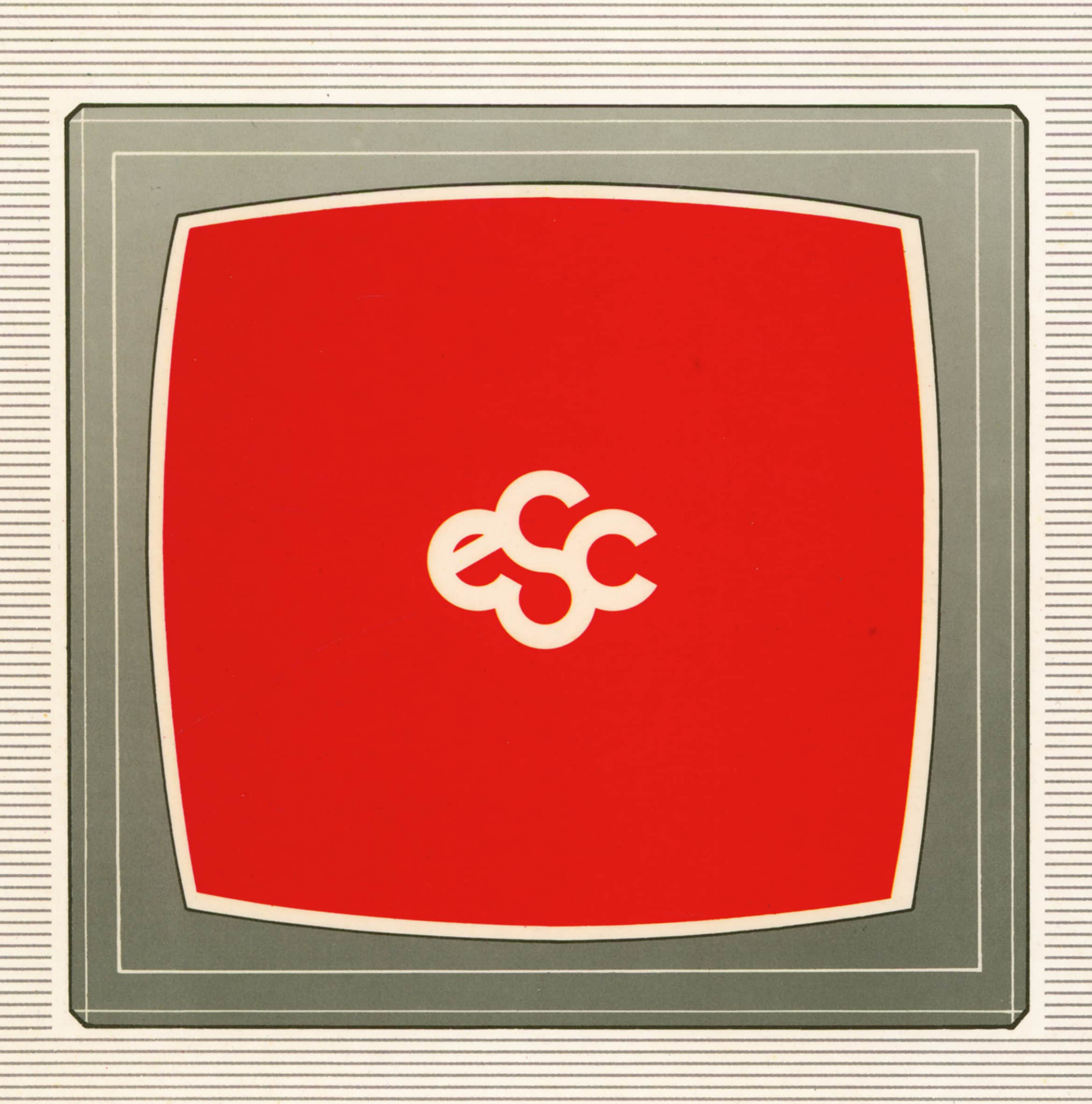
Es este un concepto que no se ha utilizado demasiado en la programación clásica, debido principalmente a problemas de implementación. Eso de que un procedimiento o función pueda llamarse a sí mismo es algo a primera vista complicado.

Hablemos algo más sobre el asunto.

Pensemos un momento en la definición de una función esencialmente recursiva, como, por ejemplo, el factorial de un número. El factorial de un número natural n (n!) sería el producto de ese número por el factorial del número menos 1 (es decir, $n! = n \times (n - 1)!$), con lo cual estamos utilizando para definir el concepto (la operación), el propio concepto. Pues bien, está claro que si deseamos hacer un programa (o función, subrutina, etc.) que realice la operación, necesitaríamos una serie de comparaciones (sentencias, IF, etc.) para implementarlo de una forma no recursiva. Excepto que el lenguaje que utilicemos permita que una función se llame a sí misma. Con lo cual la función en alguna parte de sus «sentencias» llamaría a sí misma decrementando el parámetro de cálculo (n - 1), y así sucesivamente, luego el camino de retorno (un montón de ejecuciones de RETURN) y conseguimos que el ordenador trabaje mucho y el programador muy poco.

En LISP esto no es sólo posible (como en lenguajes clásicos como MODULA-2 o ADA), sino que el propio lenguaje se ha hecho pensando en ello. Podríamos decir que es fuertemente recomendable y que ha sido implementado eficientemente.

En el próximo capítulo seguiremos explicando los fundamentos del LISP y empezaremos a adentrarnos algo en su sintaxis.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼